Il manuale MSX
P. Hoffman

Il manuale MSX

Indice

Parte Terza — L'MSX-DOS

Prefazione

Alcuni anni fa, alcuni grandi costruttori giapponesi, ai quali si unì anche la Philips, commissionarono alla Microsoft di Seattle un sistema operativo e un BASIC da installare su una nuova generazione di home computer. Questa nuova generazione doveva costituire la risposta giapponese al successo dei vari VIC-20, Commodore 64 e ZX Spectrum che a qual tempo dominavano, e in parte dominano tuttora, il mercato.
Le caratteristiche del nuovo sistema dovevano essere: espandibilità verso l'alto, portabilità totale da un computer all'altro anche di marca differente, gestione avanzata della grafica e del suono.
Il compito fu affrontato con impegno dalla Microsoft che aveva appena concluso l'affare del secolo, ovvero l'adozione del proprio sistema operativo MS-DOS da parte dell'IBM per il suo Personal Computer. Fu facile, comunque, realizzare un sistema operativo potente e versatile dalle stesse radici dalle quali era nato l'MS-DOS. L'investimento per la realizzazione del sistema non era certo esiguo e in poco tempo venne rilasciato l'MSX-DOS e il relativo MSX-BASIC, una versione molto simile al BASIC Microsoft.
I presupposti per un grande successo c'erano tutti: un solido e diversificato hardware basato sul potente Z80, un potente software di base e, grazie alla scelta della standardizzazione, l'immediata disponibilità di una grande libreria di programmi applicativi e di giochi. Il tutto è stato propugnato con la consueta forza e aggressività delle marche nipponiche e l'MSX è diventato di fatto lo standard per gli home computer, adottato da tutti i costruttori giapponesi e europei (Philips, Sony, Yamaha, Toshiba, Canon, Panasonic, Spectravideo, Yashica e moltissimi altri, per lo più non ancora importati in Italia).
Il successo di questa formula è dovuto a diversi fattori:

- Maturità del mercato. Dopo il grande boom degli anni scorsi, una vera corsa all'oro, quando i vari contendenti si spartivano il mercato a colpi di incompatibilità hardware e software, modelli sempre più potenti e prezzi sempre più bassi, è arrivato il momento della resa dei conti e molti costruttori hanno dovuto abbandonare il mercato. La dissennata guerra dei prezzi ha reso sempre più lontano l'obiettivo del recupero dei cospicui investimenti per lo sviluppo di un sistema e, d'altra parte, ogni costruttore controlla una quota di mercato così marginale da non poter imporre granché ad una clientela sempre più evoluta che vuole un prodotto più affidabile, più potente e con la massima disponibilità di software a prezzi contenuti.
- Solide radici. Il sistema MSX è un prodotto Microsoft, la casa che ha creato l'MS-DOS, che si è basata, come abbiamo visto, proprio sul potente sistema operativo creato per i personal IBM e compatibili per creare l'MSX-DOS, un sottoinsieme che però non ha nulla da invidiare al fratello maggiore. Come linguaggio è stato facile adattare il Microsoft BASIC, o MBASIC, per ottenere l'MSX-BASIC.
- Facilità d'uso. Il sistema MSX è assolutamente immediato, basta accendere il computer e si può già lavorare. Se poi si hanno già conoscenze di BASIC e di MS-DOS non ci sono assolutamente problemi. Anche per chi è alle prime armi è facile imparare ad usarlo e se ci si trova alle prese con un MSX di altra marca, ad esempio a casa di amici, non esiste la necessità di adeguarsi ad una sintassi diversa.
- Espandibilità. Il sistema MSX può crescere attorno all'unità centrale (una semplice tastiera) con una serie infinita di periferiche (monitor, stampanti, joystick, plotter, drive per floppy disk, tastiere musicali, e così via). Non ci sono vincoli di marca e si ha a disposizione un catalogo software praticamente infinito.

Parte Prima

Il computer MSX

Capitolo

Descrizione di un computer MSX

1

Su un computer MSX è facile giocare e svolgere molti lavori, anche se non si sa assolutamente nulla di cosa ci sia dentro o di come funzioni. Comunque, capirete presto che è molto più facile fare ciò che realmente si vuole, se si conoscono i principi di funzionamento del computer.
Questo capitolo descrive alcune parti dell'MSX senza entrare in dettagli tecnici. Quando lo avrete terminato, conoscerete in modo discreto cosa c'è nel vostro computer e come esso esegue un programma.
Poiché un computer MSX ha caratteristiche simili a molti altri home e personal computer, descriveremo i computer in generale, usando il sistema MSX come esempio specifico. Come ricordato nell'introduzione, tutti i computer MSX hanno moltissimo in comune, ma hanno anche alcune differenze nelle prestazioni e nell'aspetto. Gli esempi contenuti in questo libro sono stati sviluppati su un sistema Sony Hit Bit.

1.1 Che cos'è un computer

Un computer ha due componenti basilari: l'hardware e il software. In parole semplici, *hardware* è la parte fisica del computer, mentre *software* indica l'insieme dei programmi che il computer può eseguire.

L'HARDWARE

Molte persone temono di non capire il funzionamento dei computer perché non conoscono l'elettronica. Questo timore è ingiustificato ed è come

Figura 1.1 L'HIT BIT 75 P, il computer MSX della Sony

temere di non essere capaci di cucinare perché non si conosce la chimica organica. Infatti, l'elettronica non vi aiuterà affatto nello studio delle nozioni base dei computer MSX.

A lato della tastiera ci sono quattro grossi tasti con frecce che puntano in alto, in basso, a sinistra e a destra. Questi si chiamano tasti di controllo del cursore, poiché si usano per muovere il cursore sullo schermo. (Il cursore localizza sullo schermo dove apparirà un carattere battuto sulla tastiera.)

Ci sono inoltre dei tasti denominati F1, F2, F3, ecc., detti tasti funzione, e altri tasti come HOME e INS che userete durante l'esecuzione dei programmi. Nella parte posteriore del vostro computer, si trovano parecchie prese per innestare cavi, simili a quelle che si trovano nel retro degli impianti stereo e dei televisori: sono chiamate *porte* del computer. C'è probabilmente anche un interruttore che accende e spegne il computer. Molti computer MSX sono forniti degli appositi cavi di collegamento con dispositivi esterni.

Alcuni computer MSX hanno delle spie luminose vicino alla tastiera, spesso vicino al tasto CAPS LOCK o all'interruttore. Queste luci sono simili agli indicatori di uno stereo: vi dicono quando qualcosa è "acceso".

Alcune altre parti che vedete potrebbero non risultarvi familiari, ma esse funzionano in un modo che è semplice da capire. Ogni computer MSX ha un connettore per cartucce, solitamente sopra la tastiera; è un rettangolo delle dimensioni di circa 10 cm per 1.5 cm e può avere un coperchietto

Figura 1.2 Un altro computer MSX: Yamaha CX5M

mobile di protezione. Molti computer MSX sono dotati di due innesti per cartucce. Possono essere inserite una o più cartucce che contengono programmi o che aumentano le prestazioni del computer.
Il computer MSX deve essere dotato di un monitor per visualizzare programmi e immagini. Se non avete un monitor dovete collegare il vostro apparato ad un apparecchio televisivo. La procedura di installazione è trattata nel manuale di uso del computer ed è pure riportata nel Capitolo 4 di questo libro.
Il computer può essere dotato di altri dispositivi hardware, le periferiche, che potrebbero essere già collegate al vostro computer: tra le più comuni sono compresi i joystick, le stampanti e i drive. Queste ed altre periferiche sono descritte nel Capitolo 3.
Il termine hardware comprende ovviamente anche la parte elettronica che si trova all'interno del computer. Più avanti imparerete quale tipo di hardware si trova dentro il vostro MSX e un po' di più circa il suo funzionamento. Comunque, non dovrete mai preoccuparvi dell'hardware interno al vostro computer, poiché non lo vedete né lo potete modificare. L'interazione con l'hardware del computer avviene esclusivamente attraverso componenti esterni, specialmente tramite la tastiera.

IL SOFTWARE

Descrivere il software è molto più difficile che descrivere l'hardware, poiché i programmi non si vedono né si toccano. Un programma è una se-

quenza di istruzioni che dicono al computer cosa deve fare. I termini "programma" e "software" sono spesso usati in modo intercambiabile. Ci sono inoltre molti tipi diversi di software: il software che controlla il computer è molto complesso, però esiste del software talmente semplice che potete imparare a scriverlo in pochi minuti. Il livello di difficoltà dipende dal tipo di lavoro che viene programmato.
Il software può essere "visto", guardando i programmi che sono stati scritti in un linguaggio di programmazione. Per esempio, in questo libro sono riportati molti listati di programmi in MSX-BASIC: potete ribattere questi programmi nel computer, ma una volta che il software si trova dentro di esso, si trasforma in minuscoli impulsi elettrici che possono essere capiti solamente dal computer.
Il computer MSX è dotato di due programmi residenti: il sistema operativo e l'MSX-BASIC. Il sistema operativo controlla l'hardware del computer e facilita l'esecuzione dell'altro software. Non preoccupatevi di capire il sistema operativo dell'MSX, dato che le sole persone che hanno bisogno di conoscerlo sono coloro che hanno intenzione di scrivere programmi complessi.
Il sistema operativo e l'MSX-BASIC sono immagazzinati su chip chiamati ROM, all'interno del computer MSX. ROM è un'abbreviazione per *Read Only Memory* (memoria a sola lettura), che è un modo complicato per dire che i programmi nei chip sono permanenti e non possono essere modificati. Se comprate del software in cartucce, come giochi o programmi per l'ufficio, quei programmi sono anch'essi immagazzinati su ROM.
Una volta imparato l'MSX-BASIC, potrete creare i vostri programmi. Naturalmente, vorrete salvare i programmi in modo da non doverli ribattere di nuovo ogni volta che riaccendete il computer. Vedrete più avanti, in questo libro, come potete immagazzinare i programmi su nastro (usando un mangianastri) o su disco (usando un drive).
Alcuni costruttori consegnano i computer MSX con del software aggiuntivo, costituito da giochi, da programmi per l'uso professionale, o da altri programmi specifici per la grafica e per la generazione di note musicali. Questo software è spesso in cartuccia, ma può essere residente nel computer, nella ROM.

1.2 All'interno del computer

Ora che sapete qualcosa di hardware e di software, siete pronti a studiare il contenuto del computer MSX. Come abbiamo detto prima, tutti i computer MSX hanno caratteristiche molti simili e si assomigliano molto. La seguente discussione tratta dei chip che si trovano dentro tutti i computer MSX.

Avete già visto che l'unità principale include la tastiera, il connettore (o i connettori) per cartucce e molte prese per le altre parti del sistema. Sebbene non possiate aprire il computer, c'è un certo numero di cose, all'interno dell'unità principale, che è meglio conoscere, poiché riguardano l'esecuzione dei programmi e le espansioni hardware.
I computer consistono in chip collegati fra loro in modo da potersi scambiare segnali elettronici. Alcuni di questi chip sono molto importanti e, fortunatamente, sono facili da descrivere. Ogni computer MSX è dotato, nell'unità principale, del seguente hardware:

- **Una CPU Z-80A** (*Central Processing Unit* - unità centrale di elaborazione). Ogni computer ha una CPU, che rappresenta il suo "cervello". Z-80A è la CPU del computer MSX; altri tipi di CPU comuni in altri computer sono la 6502, la 8088 e la 68000. La CPU dell'MSX è in grado di elaborare 3.58 milioni di istruzioni al secondo, più velocemente, quindi, di molti altri personal computer.
- **Una memoria RAM.** Abbiamo detto prima che la ROM immagazzina il sistema operativo dell'MSX e l'MSX-BASIC e che non vi si può scrivere; non potete cioè cambiare quello che è immagazzinato nella ROM, che quindi non serve per i vostri programmi. Diversamente, la RAM è una memoria a doppio uso, poiché potete leggerla (eseguire un programma) e scrivere su di essa (modificare il programma). Comunque ogni informazione immagazzinata nella RAM viene "dimenticata" quando spegnete il computer.
  Più RAM avete, più estesi possono essere i programmi. Il vostro computer ha probabilmente una RAM da 16 K, ma può essere dotato di più RAM fino a 64 K. Un K consiste in 1024 unità di memoria, così più K avete, più memoria avete.
- **Chip speciali per la grafica.** Le immagini che il computer genera sullo schermo sono controllate da un chip speciale: questo permette alla CPU di dedicare più del suo tempo al programma. Avere un chip grafico speciale permette una maggiore velocità di esecuzione dei programmi.
- **Chip speciali per la generazione dei suoni.** Come il chip grafico, il chip sonoro è un chip "dedicato"; esso permette al computer MSX di produrre facilmente suoni complessi.
  La maggior parte degli altri chip e collegamenti nell'unità principale aiutano quelli qui elencati a svolgere i loro compiti.

1.3 L'MSX-DOS

Alcuni computer MSX hanno un drive incorporato, mentre altri ne utilizzano uno esterno che va comprato separatamente. I vantaggi di possede-

re un drive includono un più rapido immagazzinamento dei programmi e un più rapido accesso per i programmi che richiedono una grande quantità di dati.
Per gestire file su disco dovete avere anche l'MSX-DOS. Questo è normalmente fornito insieme ai drive destinati ai computer MSX, e consiste in un dischetto (vedi Capitoli 17, 18 e 19).
Per usare l'MSX-DOS, dovete avere 64 K di RAM. Ciò perché l'MSX-DOS come altri programmi, richiede una certa quantità di memoria riservata. Se il vostro computer ha meno di 64 K di RAM, potete espandere quella che avete comprando delle cartucce RAM aggiuntive (vedi Capitolo 3).
Per usare l'MSX-DOS, dovete anche poter collegare il drive al computer; ciò può presentare un problema, dato che la maggior parte dei drive si collega attraverso il connettore per cartucce. Se avete un solo connettore non potete collegare contemporaneamente il drive e la cartuccia. Molti computer MSX hanno più di un connettore o hanno scatole di espansione che permettono di aumentarne il numero.

Capitolo

Applicazioni dei sistemi

Il software disponibile per i computer MSX è estremamente vario e copre le più diverse esigenze. Siccome ogni giorno vengono scritti nuovi programmi ed i vecchi vengono costantemente migliorati, sono qui elencati solo pochi programmi specifici. Potete trovare elenchi di programmi MSX nelle riviste specializzate sugli home computer. Alcune grandi società di distribuzione, inoltre, annoverano nei loro cataloghi molti prodotti software per l'MSX.
La maggior parte del software per l'MSX è in cartuccia, dato che ogni computer MSX ha almeno un connettore per cartucce. Le cartucce rappresentano un modo conveniente per distribuire software poiché sono estremamente affidabili ed è molto difficile farne copie illegali. Alcuni produttori di software distribuiscono programmi su cassette o dischetti. Quando comprate del software, assicuratevi che sia in una forma che potete usare. Per esempio, se non avete un registratore a cassette collegato al vostro computer MSX, assicuratevi di non comprare un programma memorizzato su una cassetta. Se avete un lettore di dischetti, assicuratevi sempre che il software sia compatibile con il vostro drive. Quando siete in dubbio, chiedete al vostro rivenditore.
Gli MSX sono usati sia a casa che in ufficio, e il software disponibile segue questa distinzione. Per esempio, c'è un certo numero di programmi di word processing: alcuni sono perfetti per l'uso domestico — per i compiti a casa e la corrispondenza in generale — mentre altri includono potenti prestazioni necessarie per usi professionali. Naturalmente, potete usare il software come meglio preferite ma dovreste, prima di comprarlo, chiedere al rivenditore quale sia l'uso principale per quel software.
Il sistema operativo dell'MSX è simile al sistema operativo CP/M e ancor

di più all'MS-DOS, e molti programmi sotto CP/M e MS-DOS sono stati convertiti per essere usati sull'MSX.

2.1 Giochi

L'uso più popolare dei computer MSX è senza dubbio il gioco: ci sono centinaia di giochi disponibili, moltissimi dei quali in cartuccia.
La maggior parte dei giochi per l'MSX può essere catalogata tra i giochi d'azione e i giochi di riflessione. I primi, i cosiddetti "arcade game" sono in genere più ripetitivi e stimolano la prontezza dei riflessi.
Se non conoscete i giochi d'azione per computer, prima di acquistarli, fissate nella vostra mente i seguenti punti:

- Non giudicate un gioco dalla sua copertina. Lo sfolgorante disegno sulla scatola può sembrare stupendo, ma ricordate che è meno costoso produrre una splendida scatola che un gioco eccitante. I buoni programmatori sono più costosi dei buoni illustratori.
- Non aspettatevi di più di quanto non abbiate visto nelle sale giochi. Se non vi è piaciuto nessun videogame, decisamente non vi piaceranno i giochi arcade reperibili per i computer MSX. Molti dei giochi arcade MSX sono concettualmente copie dei videogame più diffusi, ma in genere più lenti e meno eccitanti.
- Preparatevi ad aggiungere un joystick al vostro computer. Il joystick è una periferica che controlla determinate azioni sullo schermo. Muovendo la leva e premendo il pulsante sul joystick si muovono oggetti o si compiono delle azioni. Molti dei giochi arcade che richiedono velocità ed agilità sono molto difficili da giocare senza un joystick. Muovere un oggetto per lo schermo usando i tasti di controllo del cursore invece del joystick può essere molto difficoltoso.
- Provate un gioco prima di comprarlo. Purtroppo buona parte dei negozi di computer non vi permetterà di giocare in negozio, ma voi chiedete lo stesso.

GIOCHI ARCADE

I giochi arcade per gli MSX non sono terribilmente eccitanti, ma sono popolari. Tra questi ricordiamo Mouser, Car Jamboree (una corsa di automobili che ha molti elementi di una gara di demolizione), Pitfall! (corse e salti nella jungla), Keystone Kapers (guardie e ladri), Hyper Olympics (simulazione sportiva) e Sparkie (un divertente zig-zag in un tipico gioco di labirinti).

Molti giochi divenuti famosi nelle sale sono stati convertiti per uso domestico, e molti titoli sono reperibili per i computer MSX.
Gli sport (come baseball, golf e tennis) rappresentano un altro soggetto molto popolare dei giochi MSX, ma sono raramente divertenti quanto praticare i veri sport.
Alcuni giochi d'abilità sono anche strumenti d'insegnamento. I programmi di simulazione di volo pongono sotto il vostro controllo i numerosi quadranti e luci di un piccolo velivolo e rispondono ai controlli come farebbe un vero aeroplano. L'obiettivo non consiste nel fare punti ma nel decollare ed atterrare senza schiantarsi.

GIOCHI DI RIFLESSIONE

Alcuni giochi di questa categoria vengono definiti "romanzi interattivi", mentre altri sono "giochi di simulazione". Sono come racconti che voi aiutate a creare perché decidete dove deve andare o cosa deve fare il personaggio principale.
I più interessanti sono di gran lunga i giochi di avventura: dovete risolvere misteri e conquistare tesori usando le parole per muovervi attraverso un ambiente generalmente ostile. Per esempio, se il programma vi dice che siete vicino ad una caverna e scegliete di entrarci, il programma vi dirà allora che cosa trovate nella caverna. Potete fare molte cose (prendere tesori, uccidere draghi, eccetera), ma l'obiettivo del gioco è risolvere un mistero o raggiungere una meta.
Gli scenari variano dai racconti medievali di cavalieri e draghi, alla fantascienza, ai tipici romanzi gialli. I giochi di questo tipo sono noti per il loro ambiente e per le trame intricate e l'interazione con il gioco è sorprendentemente realistica.
I giochi d'avventura in genere presentano sullo schermo un testo che descrive l'ambiente; altri visualizzano le scene: saloni di stile gotico, ponti levatoi, pipistrelli.
Altri giochi di riflessione reperibili sono le versioni di popolari giochi da tavolo. Per esempio, esistono le versioni MSX di Othello, backgammon e mah-jongg. Si possono trovare inoltre molti programmi di giochi con le carte.

2.2 Word processing

I programmi di word processing vi permettono di scrivere e salvare qualsiasi tipo di testo — appunti, lettere, rapporti e libri — e stamparlo su una stampante. La maggior parte delle persone trova che il word proces-

sing sia il programma applicativo più utile, poiché rende lo scrivere più facile; una volta presa confidenza, troverete anche voi che il word processing è uno strumento estremamente valido. Anche se avete comprato il computer per altri scopi, dovreste comunque avere un buon word processor.
Man mano che i programmi di word processing diventano più comuni, sempre più persone si rendono conto della enorme differenza che esiste tra l'usare un word processor e una macchina da scrivere: con un word processor, correggere un testo è estremamente più facile, ristrutturare una lettera o un appunto è molto rapido, e salvare il testo su un dischetto o su una cassetta del computer significa che non avrete mai più bisogno di ribatterlo.

L'ESSENZA DEL WORD PROCESSING

Un word processor ha molte analogie con una macchina da scrivere con memoria. Il programma mostra i caratteri sul video come voi li battete dalla tastiera e li salva sul dischetto o sul nastro. Inoltre potete dare un comando per stampare quei caratteri sulla stampante.
Comunque, la somiglianza finisce quando volete cambiare qualcosa che avete battuto. Con un word processor potete facilmente aggiungere o cancellare parole, frasi e paragrafi. Potete anche spostare frasi e paragrafi lungo il testo che avete creato. Ovviamente, oltre a fare questi cambiamenti, il programma vi permette di correggere il testo salvato. Un'altra prestazione di un word processor consiste nel poter decidere che il vostro documento abbia un particolare formato (numerazione automatica delle pagine, inizio del paragrafo rientrato dal margine, caratteri corsivi) quando lo stampate su carta.
Ad un programma di word processing potete dare dei comandi per dirgli cosa deve fare e dove. Con la maggior parte dei programmi si usa una combinazione di tasti per dare questi comandi (come per muoversi avanti o indietro nel testo, per cancellare, per inserire e così via). I computer MSX hanno tasti speciali che rendono l'elaborazione del testo più facile. Voi iniziate ad usare un word processor dicendo al programma quale testo sul dischetto o sulla cassetta volete editare o se volete cominciare a scrivere un nuovo testo. Lettere, capitoli o articoli, sono conservati nei file, che sono descritti in dettaglio nel Capitolo 18. Se volete scrivere un nuovo file, dite al programma che volete inserire un testo e iniziate a batterlo. Se avete intenzione di correggere un testo che avete già scritto, dite al programma il nome del file e dove si trova nel testo quello che volete correggere (come la prima linea o la seconda frase nel quarto paragrafo). Quando il programma vi porta al punto desiderato, potete quindi apportare le modifiche. In questo modo, vi potete muovere attraverso l'intero

file, cambiando il vecchio testo o aggiungendone del nuovo. Una delle più piacevoli prestazioni è che potete muovervi attraverso il file avanti e indietro come volete, non costretti ad andare solo dall'inizio verso la fine.
Quando avete terminato di correggere o di inserire un file, dite al programma di salvare il vostro nuovo file (con tutte le vostre modifiche) su un dischetto o su una cassetta. Se volete, potete stampare questo nuovo file. Dopo aver usato un word processor, non vorrete mai più tornare ad usare una macchina da scrivere!
Generalmente, più potente è il word processor, più cose potete fare con i suoi comandi di formato: alcuni permettono semplici operazioni come regolare i margini destro e sinistro, numerare le pagine, o iniziare i paragrafi lasciando uno spazio dal margine, mentre altri sono più sofisticati. Alcuni comandi, per esempio, possono eseguire funzioni come stampare automaticamente una tabella, centrare i titoli delle varie parti, porre una intestazione particolare in cima ad ogni pagina, o stampare automaticamente in grassetto o sottolineato un gruppo di parole.

COSA CERCARE IN UN WORD PROCESSOR

I programmi di word processing si possono trovare in una vasta gamma di prezzi. Stranamente, non è detto che un programma di word processing più costoso, sia più facile da usare ed abbia maggiori capacità di una versione più a buon mercato. I criteri più importanti nella scelta sono che il programma sia facile da imparare ed usare e che abbia prestazioni sufficienti per soddisfare le vostre necessità.
Quelle che seguono sono alcune domande che dovete porvi quando iniziate la vostra ricerca del miglior programma:

- Quanto è facile inserire, cambiare, formattare e stampare il testo? Voi probabilmente volete evitare programmi che richiedono di tenere a mente complicate sequenze di tasti da premere per dare comandi di edit. Ciò è particolarmente vero se intendete usare un word processor solamente per applicazioni non professionali.
- Il word processor può funzionare insieme alla vostra stampante? Alcune stampanti possono selezionare vari stili di stampa (chiamati font), usare una spaziatura proporzionale o creare esponenti e deponenti. Potete utilizzare queste capacità solo se il vostro word processor è stato configurato per il tipo di stampante che avete; in questo caso la stampante potrà produrre gli effetti comandati dal programma di word processing. La maggior parte dei word processor sono configurati soltanto per pochi tipi di stampanti.
- Il programma comprende la possibilità di creare un'agenda di indirizzi? Molti programmi vi permettono di fondere un file agenda con una

lettera per crearne una serie, cambiando il nome e l'indirizzo in cima ad ogni lettera.

- Il programma ha prestazioni per utenti esperti? Molte persone che usano regolarmente un word processor vogliono programmi con prestazioni avanzate che, sebbene troppo complicate per i principianti, permettono molta più flessibilità. Per esempio, alcuni programmi permettono di impostare l'esecuzione di parecchi comandi in sequenza, senza ulteriori interventi da parte dell'utente.
- Quanto è facile da imparare? Alcuni programmi sono corredati di una documentazione ben scritta, di menu di aiuto sul video, di mascherine o di tabelle di consultazione dei comandi per aiutarvi ad apprenderne l'uso il più presto possibile. Altri programmi hanno documentazioni povere, scomodi tasti per i comandi, e altre caratteristiche che ne impediscono un uso immediato.

Alcuni word processor più completi includono un programma di controllo ortografico. Questo programma esamina ogni parola nel file del vostro testo, guarda nel suo vocabolario e segnala quelle che non riconosce. Poi vi lascia decidere se avete commesso un errore di ortografia o se semplicemente il programma non conosce la parola (per esempio una sigla o un nome proprio). Come potete immaginare, ogni file può avere molte parole che non si trovano nel vocabolario del programma, per cui è particolarmente importante che il programma di controllo ortografico vi permetta di aggiornare facilmente il vocabolario.
Molti dei programmi di word processing per l'MSX sono per uso professionale, ma ne esistono anche per uso domestico più semplici da imparare poiché hanno minori prestazioni di quelli per l'ufficio. Spesso si può imparare ad usare un semplice word processor in meno di un'ora. I programmi di word processing domestici spesso utilizzano le caratteristiche grafiche e sonore dell'MSX in modo più creativo dei programmi per l'ufficio. Per esempio il programma Bank Street Writer è un programma famoso per la semplicità d'uso ed è consigliato anche per i bambini.

2.3 Comunicazioni

Un componente hardware che spesso si collega ai computer MSX è il modem. Un modem vi permette di comunicare con altri computer attraverso la normale rete telefonica. I modem fanno sì che possiate usufruire della grande quantità di servizi di informazione (tanto vasti quanto lo sono i data base) disponibili su grandi computer.
Come funziona un modem? E molto semplice. Con i modem, le linee della società telefonica si trasformano in una sconfinata rete tra i computer.

Come il vostro computer trasmette i caratteri alla stampante attraverso un cavo, un modem trasmette caratteri a un altro modem di un altro computer attraverso la linea telefonica. Sfortunatamente la maggior parte dei modem non vi permette di comunicare molto rapidamente, potendo inviare 30 caratteri al secondo (300 Baud) o 120 caratteri al secondo (1200 Baud).
Per scambiare dati con altri computer è necessario più di un modem e avrete bisogno di programmi per comunicazione che dicano al modem come inviare le informazioni. Quelli che seguono sono i compiti che i programmi per comunicazione sono in grado di svolgere:

- Un programma per comunicazione collega il vostro computer ad un altro: questa operazione è chiamata "simulazione di terminale". Per esempio, se vi collegate con una unità centrale, i programmi per comunicazione trasmettono tutti i caratteri da voi battuti all'altro computer e stampano sul vostro schermo tutti i caratteri inviati dall'altro computer.
- I programmi per comunicare vi permettono di trasmettere testi tra computer forniti di modem. Trasmettere un testo con un programma per comunicazione è più rapido e più sicuro che stampare il file e inviarlo per posta.
- Alcuni programmi permettono di trasmettere software attraverso le linee telefoniche. Per far questo, è necessario che entrambi gli utenti facciano girare lo stesso programma per comunicazioni o che abbiano due programmi che usano gli stessi metodi di trasmissione e ricezione. Siccome i programmi non sono altro che file con caratteri speciali, la maggior parte dei programmi per comunicazioni per la trasmissione e la ricezione di testi permette anche di trasmettere e ricevere programmi.
- Alcuni modem sono in grado di comporre i numeri telefonici al vostro posto. Questa composizione automatica dei numeri necessita di un programma che impartisca i comandi giusti. Molti dei programmi per comunicazione vi permettono di memorizzare i numeri telefonici per poi farli comporre dal programma stesso. Ciò rende l'uso del modem ancora più semplice: voi dovete semplicemente dire al vostro programma per comunicazioni di comporre il numero di un certo computer.

Quasi tutti i programmi per comunicazioni possono eseguire una simulazione di terminale e un trasferimento di testi e buona parte di essi l'autocomposizione dei numeri e il trasferimento dei programmi.
Le altre caratteristiche che dovreste cercare in un software per comunicazioni sono la facilità d'uso (sono tutti piuttosto semplici) e la compatibilità con gli altri programmi. Dovreste cercare di sapere quale "protocollo di trasmissione" (il metodo usato dai computer per controllare l'esattezza dei dati inviati) viene utilizzato dal computer con cui comuni-

cate, poiché il vostro programma deve usare lo stesso protocollo. Due dei protocolli più comuni sono lo XON/XOFF conosciuto anche come "Control-Q/Control-S", e il MODEM7, usato quasi esclusivamente dagli home computer.

2.4 Fogli elettronici

I fogli elettronici hanno una gran parte del merito di aver reso popolari i personal negli uffici. Finché il VisiCalc non fu lanciato sul mercato, gli uffici usavano i personal computer quasi esclusivamente per il word processing e la contabilità. Il personal divenne uno strumento molto più utile per gli uffici quando fu sviluppato questo programma in grado di seguire gli sviluppi di una società.
Negli anni in cui fu introdotto il VisiCalc, molte persone scoprirono utilizzazioni per il foglio elettronico anche per la casa, per stendere la dichiarazione dei redditi o pianificare il bilancio familiare. Esistono molti programmi di foglio elettronico per l'MSX, come il Multiplan della Microsoft.
L'idea base che sta dietro a un foglio elettronico è molto semplice: lo schermo è diviso in celle rettangolari, simili alle caselle del foglio di bilancio di un contabile (vedi Figura 2.1). Le colonne e le righe sono identificate con numeri. Le celle vengono individuate con i numeri della riga e della colonna nelle quali si trovano. Per esempio il numero 27.700 è nella cella R11C4.

	1	2	3	4	5	6
1	ANNI	-1982-	-1983-	-1984-	-1985-	-1986-
2						(stima)
3						
4	Vendite hardware	10.425	12.340	15.900	23.770	22.230
5						
6	Vendite software	5.890	6.210	8.450	13.440	16.730
7						
8	Affitti	0	0	3.350	6.210	8.920
9						
10						
11	Totale entrate	16.315	18.550	27.700	43.420	47.880
12						

Figura 2.1 Esempio di foglio elettronico

Ogni cella può contenere tre tipi di informazioni: numeri, equazioni, testo.

- **Numeri.** Può essere una somma di denaro incassata, il numero di pezzi contenuti in un magazzino, il numero di ore lavorate nel mese, o qualsiasi altro possibile numero. Questi numeri sono i dati che voi inserite. Per esempio, nella Figura 2.1, il numero della cella R4C2 rappresenta il ricavo nel 1982 per la vendita di hardware.
- **Equazioni.** Le equazioni pongono in relazione le celle tra di loro. Per esempio, la cella R11C2 nella Figura 2.1 contiene l'equazione R4C2+R6C2+R8C2. Sommando il contenuto di queste tre celle, si ottiene il totale delle entrate del 1982. Allo stesso modo le celle della colonna 6 contengono le equazioni per la proiezione delle vendite per il 1986, sulla base dell'incremento delle vendite negli anni precedenti. Le equazioni non vengono mostrate sul foglio elettronico, sebbene possiate vederle posizionando il cursore sulla cella. Diversamente, viene mostrato solo il risultato dell'equazione.
  Le equazioni possono utilizzare anche altre funzioni matematiche come la moltiplicazione e la divisione. Per esempio una cella può contenere un'equazione che determina in percentuale il contributo dato da un settore al guadagno dell'intera società.
- **Testo.** Il testo è usato per denominare le varie parti del foglio elettronico, per ricordare cosa rappresentano i numeri; un esempio è la parola "Affitti" nella cella R8C1.

Il bello dei fogli elettronici è che vi permettono di cambiare facilmente i numeri e le equazioni e poi vedere i risultati, realizzando calcoli "ipotetici". Per esempio, se volete vedere cosa succederebbe se il piano di ricavi mostrato in Figura 2.1 fosse modificato con un incremento degli affitti, potete cambiare i numeri della colonna del 1985 e vedere come ciò influenza la colonna del 1986. Oppure potreste cambiare le equazioni usate nella colonna del 1986 e vedere come cambia la proiezione dei ricavi.
I programmi di foglio elettronico hanno molte applicazioni come strumenti multiuso per l'elaborazione numerica. Oltre a fornirvi un semplice sistema per fare ipotesi sul futuro, vi offrono un sofisticato metodo per seguire i movimenti di capitale di una società o per controllare il vostro conto corrente personale. Dopo aver esaminato i vari modi con i quali il denaro entra ed esce dalla società, un dirigente può considerare le possibili strategie per massimizzare i profitti. Potete anche usare i fogli elettronici per elaborare progetti di ingegneria o per quasi tutti quei lavori che richiedono equazioni.

2.5 Programmi di gestione di data base

Un programma per la gestione di un data base organizza le informazioni, estrae i dati richiesti, vi permette di aggiornare e di consultare i dati già presenti nel computer. Nell'ufficio un data base vi aiuta nel controllo dell'archivio, del registro generale, dei nomi e degli indirizzi dei clienti e così via. A casa vi può aiutare ad organizzare i conti, i passatempi o altri interessi personali.
Alcuni data base per MSX sono stati concepiti per l'uso domestico e sono quindi molto semplici. Altri sono per l'ufficio e possono essere piuttosto complessi per le loro potenti prestazioni.
Un data base è una raccolta di informazioni che viene memorizzata nel computer. Ogni data base ha una struttura che è il modello o formato base sul quale saranno memorizzate le informazioni. La struttura determina il tipo di informazioni che potete memorizzare (come nomi, indirizzi, parti numeriche e così via) e se le informazioni sono dei testi o valori numerici. Per esempio, se tenete un archivio di clienti, la struttura del data base potrebbe assomigliare alla Figura 2.2.
All'interno della struttura, i dati sono contenuti in campi e record. I campi sono informazioni che hanno un significato particolare nell'insieme dei dati. Per esempio sono campi un nome, un indirizzo e così via. La struttura definisce le caratteristiche di ogni campo.
Un gruppo di campi tra loro correlati, viene chiamato record ed ogni re-

Campo	**Tipo**	**Commenti**
Codice cliente	Numerico	4 cifre
Società	Testo	Max. 20 caratteri
Contatto	Testo	Max. 25 caratteri
Indirizzo	Testo	Max. 25 caratteri
CAP	Numerico	5 cifre
Città	Testo	Max. 15 caratteri
Provincia	Testo	2 caratteri
Telefono	Testo	Nella forma XXXX-XXXXXXX
Rappresentante	Testo	Max. 25 caratteri
Saldo	Numerico	In migliaia di lire

Figura 2.2 Struttura di un data base di clienti

Codice cliente	4120
Società	Cartoleria Rossi
Contatto	Vittorio Bassi
Indirizzo	Via Pastrengo 142
CAP	20146
Città	Milano
Provincia	MI
Telefono	__02-7058804
Rappresentante	Andrea Cortese
Saldo	1.550

Figura 2.3 Un record del data base clienti

cord ha una serie di valori per ogni suo campo. Una valida analogia è il catalogo di una biblioteca. L'indice rappresenta l'insieme dei dati, ogni scheda rappresenta un record ed ogni informazione (come il nome dell'autore o il titolo del libro) rappresenta un campo.
La Figura 2.3 mostra un record di un data base definito secondo il modello della Figura 2.2.
Ogni campo appare solo una volta nel record. Non è necessario che riempiate tutti i campi di un record. La Figura 2.4 mostra una visione schematica di un data base. Per esempio, poniamo che la Cartoleria Rossi cambi numero di telefono. Usate il vostro data base per trovare il giusto record e quindi cambiate il numero telefonico elencato tra i campi.
Sostituire ed aggiungere record è utile, ma la principale ragione di usare un data base è per estrarre informazioni dall'archivio.

	Codice cliente	Società	Contatto	Indirizzo	...
Record	4120	Cartoleria Rossi	Vittorio Bassi	Via Pastrengo 142	...
Record	1107	Libreria Unità	Marco Paoloni	Via Meda 25	...

Campi

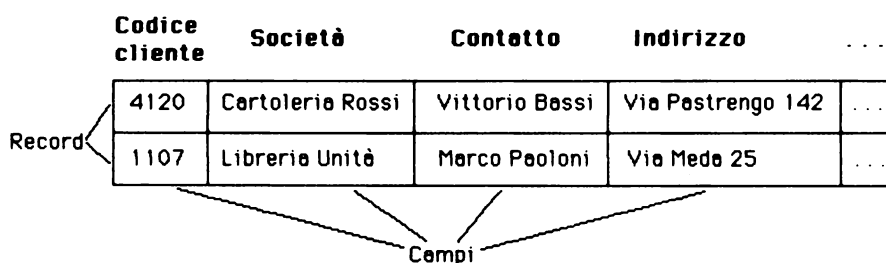



Figura 2.4 Visione schematica di un data base

Un buon data base permette di visionare le informazioni in base a qualsiasi criterio voi stabiliate. Alcune richieste che potete fare al vostro data base sono:

- Fare una lista di tutte le società che hanno un CAP che inizia con 201.
- Stampare etichette con gli indirizzi di tutti i clienti di Franco Tonini.
- Fare una lista di tutti i clienti che devono denaro, ordinandoli in base alla somma dovuta e mostrando la somma totale dovuta.
- Fare una lista di tutti i clienti aggiunti all'archivio nell'ultimo mese.

Un data base è molto utile sia per eseguire un'analisi delle informazioni, che per dare totali di quantità numeriche. Un altro compito che può svolgere bene è vagliare le informazioni. Per esempio voi potreste voler esaminare il record di una società, ma tutto quello che ricordate è che il suo nome inizia per "P" e che si trova a Roma. Il vostro data base può ricercare i record che corrispondono contemporaneamente a queste due caratteristiche.
Potete ben immaginare che un data base può impiegare molto tempo per la ricerca di un'informazione in un file di dati molto grande. Per accelerare la ricerca, quasi tutti i data base in commercio utilizzano dei campi chiave. Quando definite la struttura, dite al data base quali campi pensate di usare più frequentemente per la ricerca (per esempio due buone scelte sono il nome della società e la località). Il data base quindi riserverà un file speciale di chiavi che sarà usato per trovare velocemente i record da voi selezionati.
Per imparare ad usare un avanzato ma complesso data base possono essere necessarie parecchie ore, mentre per uno semplice occorreranno solo pochi minuti. Se comprate un data base siate certi di prenderne uno che soddisfi le vostre necessità (e naturalmente il vostro portafoglio).

2.6 Bilancio familiare

Gestire il vostro bilancio familiare è naturalmente più facile che gestire un intero giro di affari. Un programma per bilancio familiare è quindi molto più semplice da usare di un programma di amministrazione per l'ufficio.
Buona parte dei programmi di bilancio familiare vi aiuta a:

- Memorizzare ogni assegno che emettete compresa la sua motivazione
- Dividere le spese per categoria
- Fare un grafico delle finanze
- Ricordare le spese periodiche

Questi compiti sono molto facili da svolgere, e molti dei programmi di bilancio per l'MSX sono capaci di altre prestazioni.
Se ve la sentite, potete scrivere i vostri programmi per il bilancio familiare in MSX-BASIC. Un programma di questo tipo può essere lungo ma non è molto difficile da fare.

2.7 Migliorare se stessi

Una delle ragioni per cui molte persone comprano un computer, è per imparare nuove cose. Tutti i loro desideri sono di solito appagati dalla soddisfazione rappresentata dall'uso di un word processor al posto di una macchina da scrivere (o di carta e penna). Comunque esistono anche molti altri programmi che vi permettono di conoscervi meglio, di migliorare la vostra abilità o di approfondire nuovi argomenti.
Ad esempio i programmi per l'addestramento alla dattilografia, disponibili per l'MSX, sono un modo eccellente per imparare a scrivere a macchina nella quiete di casa. Molti di questi programmi non solo insegnano le basi dell'uso della tastiera, ma anche come incrementare la velocità; alcuni hanno anche giochi che rendono divertente l'apprendimento.
Ci sono inoltre molti corsi scolastici che coprono tutte le esigenze, dall'asilo all'età adulta. Non tutti gli insegnanti sono d'accordo sull'utilità del computer per l'insegnamento ai bambini di materie basilari come la lettura e la matematica; alcuni lo trovano eccellente, altri lo accusano di trasformare l'insegnamento in un esercizio meccanico.
Ci sono inoltre giochi educativi che esercitano sia nei bambini che negli adulti, la capacità di coordinamento e di memorizzazione. Sebbene questi programmi siano spesso presentati come giochi, essi sono in realtà programmi che fanno acquisire capacità basilari nella vita quotidiana.

2.8 Studiare i computer

È molto diffusa nella società l'idea che se non si conoscono i computer si è destinati a svolgere un lavoro umile. Questo timore è completamente infondato.
Ciò che leggete nei Capitoli dall'1 al 4 di questo libro è probabilmente più che sufficiente per una cultura di base sui computer, che vi permetterà di non avere paura di loro se ne incontrerete nel vostro lavoro. Certamente non avete bisogno di sapere come programmare un computer o come funzionano i componenti elettronici al suo interno, per avere un posto nella società moderna.

Nella maggior parte dei casi, quello che avrete imparato sul vostro computer MSX, sarà largamente applicabile agli altri computer, sia che si tratti di home computer che di grandi unità centrali. Quindi potete usare il vostro computer MSX come il primo mattone per imparare di più sugli altri computer.
Una delle prime cose da fare con il computer MSX è imparare l'MSX-BASIC. Imparare il linguaggio vi può far capire molte cose: come il computer svolge i suoi compiti; quanto sia complicato progettare un grande programma come un word processor; le relazioni tra le differenti parti del computer (come la tastiera e la CPU).
Ci sono altri linguaggi disponibili per i computer MSX che potrebbero interessarvi, ad esempio il linguaggio C e LOGO. Il linguaggio C è un linguaggio di programmazione avanzato. LOGO è molto diffuso per l'insegnamento della programmazione ai bambini.
È importante ricordare che non avete bisogno di imparare nulla sui linguaggi di programmazione per usare il vostro computer, anche se la padronanza di un linguaggio potrà esservi utile in mille occasioni. Non preoccupatevi se in principio trovate che l'MSX-BASIC vi spaventa: succede a molte persone.
Se volete imparare di più sull'hardware, ci sono molti libri sul mercato che descrivono la struttura del computer. Comunque è estremamente difficile descrivere l'hardware del computer senza essere molto tecnici e sono perciò necessarie conoscenze di elettronica per capire come i vari chip interagiscono all'interno del vostro computer.

Capitolo

Come espandere il computer MSX

Nel capitolo precedente abbiamo esaminato la grande varietà di programmi disponibili per l'MSX. Questo capitolo tratta dell'hardware che potete aggiungere al vostro sistema MSX. Alcuni dei prodotti menzionati in questo capitolo sono di serie per alcuni modelli mentre altri sono disponibili come optional. Il connettore per cartucce standard favorisce l'acquisto sia del software che dell'hardware per il computer MSX: le fabbriche di periferiche non si devono preoccupare infatti del tipo di prese presenti sui diversi MSX; devono semplicemente costruire una cartuccia che agisca da interfaccia per il loro hardware.
Poiché un computer MSX è relativamente economico, vi potreste chiedere perché comprare dell'hardware che in alcuni casi costa di più del calcolatore. Alcune periferiche sono molto utili per la casa o per il lavoro ed esse permettono all'MSX di eseguire più funzioni (come il word processing). Potreste comprare altre periferiche semplicemente perché vi affascinano. Come per il software, anche nel caso dell'hardware ce n'è molto di più di quanto ogni singolo utente possa volere o avere bisogno.
Questo capitolo tratterà di molte periferiche disponibili per il computer MSX: via via che l'MSX diventerà più popolare verranno progettate nuove periferiche.

3.1 Connettori per cartucce

Se il vostro MSX ha un solo connettore per cartucce, la prima cosa che vorrete acquistare sarà una espansione di connettori. Questo apparecchio

occupa il vostro connettore e, come una presa multipla vi mette a disposizione due o più nuovi innesti per cartucce.
Alcuni computer MSX includono anche una porta che non è un innesto standard MSX. I produttori di questi computer spesso vendono connettori multipli di cartucce MSX che si innestano in queste porte mettendo a disposizione più connettori per cartucce senza perderne alcuno. Per usare questo connettore comunque, dovete comprare l'espansione costruita dallo stesso produttore del vostro MSX. Prima di acquistare periferiche che richiedano un innesto nel connettore per cartucce, considerate se ne avete abbastanza a disposizione. Per esempio se il vostro computer ha un solo innesto e volete usare un word processor su cartuccia assieme ad una stampante che deve essere collegata attraverso il connettore, non potrete usarli contemporaneamente senza comprare un connettore multiplo. Un'alternativa è comprare una stampante che si colleghi alla relativa porta che la maggior parte dei computer MSX (ma non tutti) ha.

3.2 RAM addizionale

Se avete un MSX con una RAM da 16 K o 32 K, potreste volere espandere la quantità di memoria. L'MSX-BASIC può usare fino a 32 K di RAM, mentre l'MSX-DOS richiede 64 K. Se siete un esperto programmatore, potreste aver bisogno di più memoria per scrivere i vostri programmi in BASIC o in altri linguaggi che usano più di 32 K di RAM. Ci sono cartucce che vi permettono di espandere la memoria dei computer MSX. Prima di comprare una qualsiasi cartuccia dovrete verificare con il vostro rivenditore che la memoria in più sia compatibile con il vostro computer e che il software in vostro possesso sia in grado di utilizzare la memoria aggiunta. Per esempio, se avete intenzione di usare il BASIC, aggiungere delle RAM in più ad un computer che ha già 32 K di memoria, non vi darà maggiori potenzialità di programmazione. Quindi siate certi di poter usare la memoria in più prima di comprarla.

3.3 I joystick

Molti dei giochi a disposizione dei computer MSX sono controllati da joystick, molto simili a quelli dei videogame da sale giochi. Un joystick vi permette di puntare velocemente verso l'alto, verso il basso, a sinistra o a destra; i joystick hanno anche un pulsante di "sparo" spesso usato per eseguire delle azioni, come sparare ad astronavi aliene.

Tutti i computer MSX hanno almeno una porta per collegare un joystick e molti ne hanno una seconda per un altro joystick. Ogni MSX può usare sia joystick costruiti per altri MSX sia joystick compatibili con i più diffusi videogame. Molti comprano joystick economici ma ci sono anche modelli più costosi per giocatori "esperti".

3.4 I registratori a cassette

Tutti i computer MSX hanno una presa per collegare un registratore a cassette. Su queste potete memorizzare e caricare programmi da voi scritti in MSX-BASIC o memorizzare dati.
Alcuni comuni mangianastri sono compatibili con i computer MSX; se ne possedete uno potete utilizzarlo, invece di comprarne un altro appositamente per il computer.
La cosa essenziale è che i mangianastri abbiano tre prese: microfono, auricolare (o altoparlante) e comando remoto dell'alimentazione. Assicuratevi che gli spinotti forniti col computer si inseriscano nelle prese del mangianastri. Se è così potrete, con ogni probabilità, usarlo con succes-

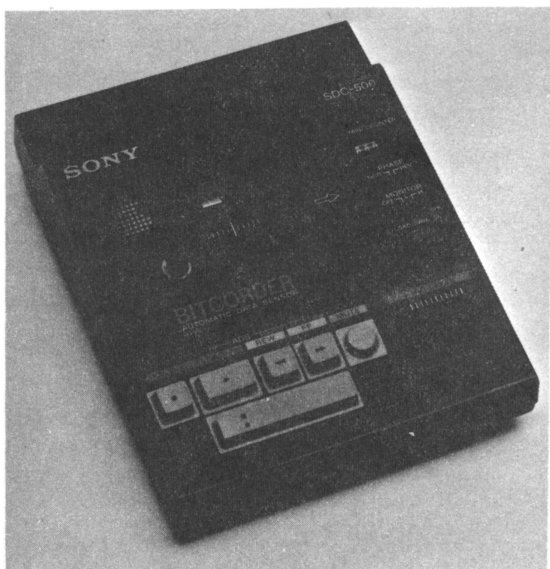



Figura 3.1 Il registratore a cassette Sony

so. Se non avete un mangianastri compatibile compratene uno economico, reperibile tanto nei negozi di computer quanto nei negozi di hi-fi.
Le cassette che potete usare per immagazzinare informazioni sono le stesse usate per registrare musica. Alcune case vendono cassette "per computer", più costose ma spesso identiche a normali cassette di alta qualità.

3.5 I drive

Se usate molto il vostro computer MSX, specialmente se lo usate per scopi professionali, avrete presto necessità di un drive. Un drive memorizza programmi e dati molto più velocemente ed in modo più semplice di un registratore a cassette.
L'unico svantaggio di un drive sta nel suo costo piuttosto elevato. Non è improbabile che costi almeno quanto un computer MSX, costo spesso proibitivo per molti utenti. A ciò si aggiunge, ad aggravare la situazione, il fatto che certi programmi che richiedono un drive, sono spesso molto costosi. Ogni differente modello di drive può immagazzinare una diversa quantità di dati, quindi è importante che, prima di comprarne uno, giriate un po' di negozi e confrontiate le prestazioni.
Quando salvate dei dati su un disco o una cassetta, le informazioni vengono immagazzinate in un file. Memorizzare informazioni in file è come ordinare dati in cartelle in un archivio. Un file è una raccolta di informazioni, identificata da un unico nome da voi assegnato. Memorizzare dati nei file è basilare per usare il computer: senza i file, il vostro lavoro andrebbe perduto quando spegnete il computer.
Le informazioni conservate nei file possono consistere in testi (ad esempio una lettera), dati (contabili o altro) o in programmi (come fogli elettronici, word processor, programmi BASIC). Il file può essere di qualunque lunghezza, limitato solamente dallo spazio disponibile su dischetto o sulla cassetta su cui è memorizzato. Quando volete che un programma elabori un file (sia per estrarre dati dal file, che per aggiungerne), dovete semplicemente usare il comando di accesso al file.
Come precedentemente detto, è molto più conveniente memorizzare file su dischetti piuttosto che su nastro. Un dischetto consiste in un supporto rotante di materiale plastico flessibile ricoperto da un materiale magnetico. I dischetti usati dai computer MSX sono i micro-floppy da 3 pollici e mezzo a singola faccia.
Cosa succede quando l'MSX recupera un file da un disco? Il disco ruota ad alta velocità mentre la testina del drive si muove avanti e indietro, diversamente da un mangianastri. Quando richiedete di vedere un file, la testina prima si sposta sul *directory*, una zona speciale del dischetto dove

si trovano informazioni su tutti file; trova poi la locazione fisica del file da voi desiderato e quindi si sposta nella zona dove è memorizzato quel file su disco.
Vi potreste chiedere come la testina del drive trovi il file. Nel directory ci sono dei parametri, due per ogni file, che identificano la traccia e il settore; l'MSX-DOS utilizza questi dati per localizzare la posizione di ogni file su dischetto. Il numero di byte per ogni settore è costante in un disco, ma i drive che potete acquistare per il vostro MSX possono avere un numero differente di byte per settore e di settori per traccia. Per fortuna, l'MSX-DOS tiene conto di tutte queste differenze. La Figura 3.2 mostra le tracce e i settori su un dischetto. Il numero delle tracce è in pratica la misura della distanza del file dal bordo del dischetto. Le tracce sono cerchi concentrici sul dischetto; poiché ogni traccia può contenere una grande quantità di dati, esse vengono suddivise in settori. I settori si contano in senso antiorario da un punto fissato, percorrendo tutta la traccia del disco fino a tornare al punto di partenza.
I dati del directory dicono quindi alla testina di quanto deve muoversi e

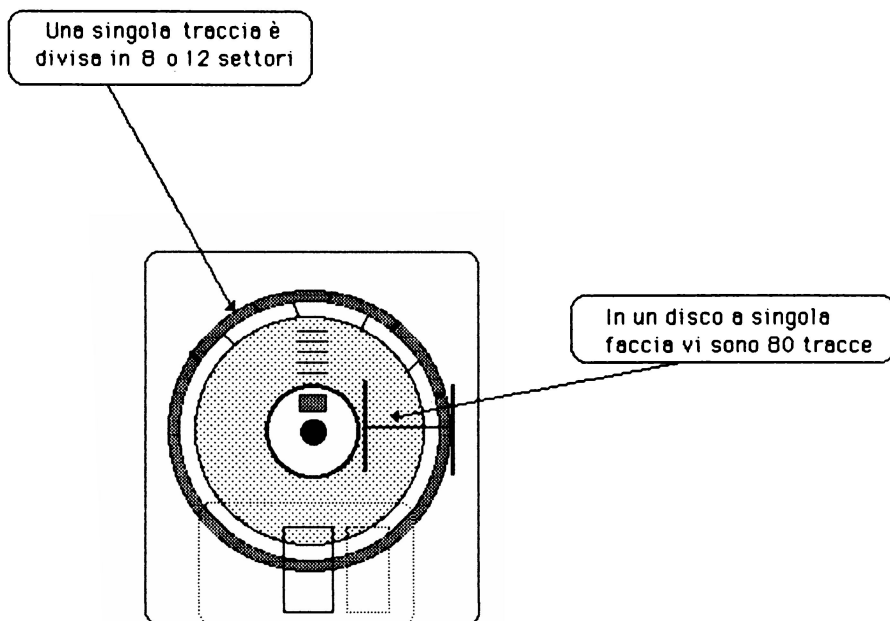



Figura 3.2 Schema di un dischetto

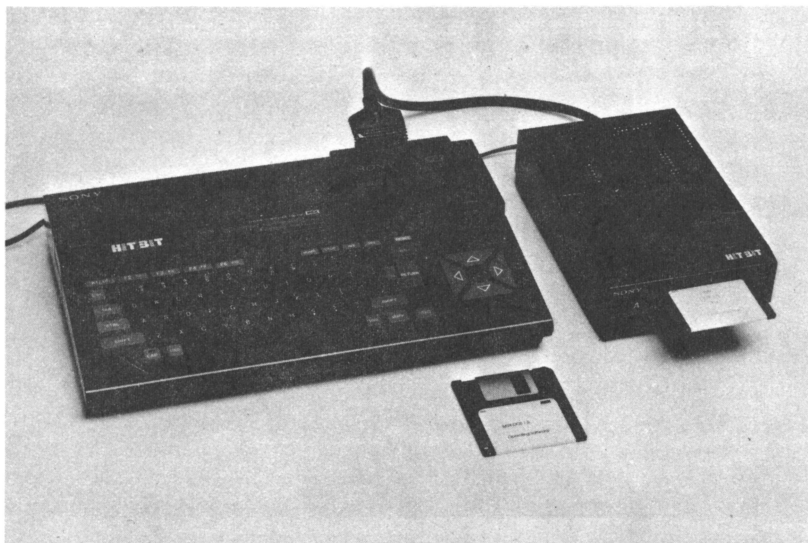



Figura 3.3 Il drive Sony HBD-50 collegato al computer

quanto deve ruotare il disco prima di iniziare a leggere il file. Trovare una traccia è come selezionare una canzone su un disco. Selezionare il settore è come cercare una o due battute musicali in una canzone.

3.6 Le stampanti

Dopo aver visto i dati sullo schermo del computer, vorrete probabilmente stamparli per consultarli in futuro o per mostrarli ad altre persone. Per stampare i dati su carta avete bisogno di una stampante: le stampanti variano molto per caratteristiche e prezzo. Una normale stampante si può collegare al computer sia attraverso l'apposita porta che attraverso il connettore per cartucce. Potete predisporre la stampante a stampare caratteri per mezzo dell'MSX-BASIC o dell'MSX-DOS. Alcune stampanti possono inoltre produrre grafica (disegni) attraverso l'esecuzione di speciali programmi.
Le stampanti si dividono in due categorie: a matrice di punti e a margherita. Le prime stampano i caratteri combinando molti punti per formare ogni lettera. Le stampanti a margherita, invece, stampano ogni carattere come le macchine da scrivere. Le stampanti a matrice sono solitamente meno costose e stampano più rapidamente di quelle a margherita, ma

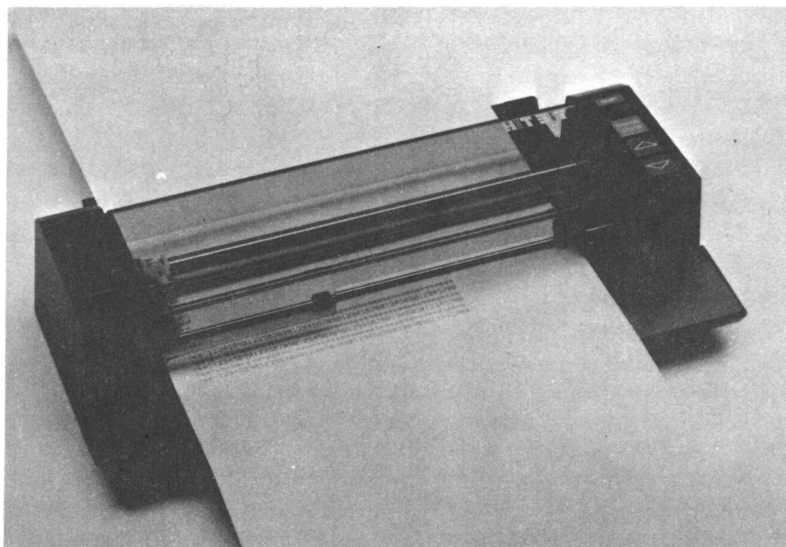



Figura 3.4 La stampante-plotter a colori Sony PRN-C41

l'aspetto delle stampe di queste ultime è decisamente migliore; esistono anche stampanti a matrice ad alta qualità. Alcune stampanti a matrice possono stampare disegni, mentre quelle a margherita non possono. Le stampanti si usano principalmente con programmi di word processing. Potete scrivere o correggere una lettera o una relazione usando il word processor e poi stamparne il risultato.

3.7 Periferiche aggiuntive

Ci sono naturalmente molte altre periferiche che potete acquistare per il vostro MSX. Questo paragrafo ne presenta alcune fra le più interessanti.

PER LA MUSICA

Una delle più diffuse applicazioni degli home computer è costituita dalla creazione di musica elettronica. I computer MSX hanno capacità di suono migliori della maggior parte degli home computer, ed è estremamente facile sfruttarle. Potete infatti comprare cartucce per il vostro computer che lo convertano in un sintetizzatore: alcuni computer MSX includono

nell'unità principale un'interfaccia per sintetizzatore e strumenti musicali. Potete controllare il sintetizzatore per mezzo dell'MSX-BASIC o della tastiera.
Molti computer MSX hanno prese che permettono di collegare il segnale sonoro ad amplificatori esterni. Vi sarà possibile quindi creare musica ed effetti sonori di alta qualità.

PER LA GRAFICA

La grafica per computer è uno dei campi in più rapido sviluppo: i prezzi delle periferiche per la grafica stanno crollando e molti dispositivi che una volta erano disponibili solo per grandi utenti, possono ora essere acquistati anche da chi usa il computer in casa.
Una periferica che sta diventando sempre più diffusa è la penna ottica, che si appoggia sullo schermo per "disegnare". Potete usare la penna ottica per fare disegni, per rispondere rapidamente a domande sullo schermo (ad esempio per segnare delle caselle) e per molti altri compiti. Anche la tavoletta grafica può essere usata come una penna ottica per disegnare. Voi disegnate sulla tavoletta con uno stilo e il vostro movimento viene riportato sullo schermo.
Se infine volete leggere direttamente fotografie o qualsiasi altro tipo di immagine, senza dover ridisegnarle, potete usare un digitalizzatore di immagini, che in pratica consiste in una fotocopiatrice e alcuni altri componenti elettronici collegabili al computer. L'immagine viene "digitalizzata" in una forma che il computer può leggere, memorizzata, e quindi modificata a piacere e stampata.
Altre periferiche vi permettono di mescolare le immagini della televisione con quelle che avete memorizzato nel vostro computer MSX. Questi apparati possono inoltre memorizzare le immagini televisive e permettono di modificarle cambiandone il colore o cancellandone parti.

PER IL CONTROLLO DELLA CASA

I computer MSX possono controllare molte apparecchiature elettriche attraverso un centro di controllo della casa.
Tra le applicazioni più semplici ci sono l'accensione e lo spegnimento delle luci a orari prescelti o l'avvio della caffettiera elettrica ogni mattina. Si può acquistare dell'hardware più sofisticato che permette al computer MSX di controllare i sistemi di sicurezza della casa per prevenire i furti e di avvertire la polizia o i vigili del fuoco in caso di emergenza.
Potete usare l'MSX anche per controllare i più diversi apparecchi, per esempio potete comperare hardware che vi permetta di controllare un

lettore di compact disk, in modo da poter saltare a una qualsiasi sezione del disco, in base ai dati memorizzati nel computer. Inoltre, come progredirà la tecnologia del laser disk, saranno moltissime le applicazioni che utilizzeranno i computer MSX insieme ai laser disk: ad esempio enciclopedie o gallerie di immagini da sfogliare con il computer.

ROBOT

Malgrado le previsioni degli ultimi cinquant'anni, i robot che puliscono la casa e preparano il pranzo non sono ancora molto diffusi. I robot che potete acquistare hanno una possibilità d'uso molto limitata e solitamente sono alquanto goffi. Comunque lo studio della robotica si è sviluppato nell'ultimo decennio, e sono disponibili alcuni piccoli robot. Alcuni robot possono essere programmati dal vostro MSX; per esempio un robot economico può avere un innesto MSX per cartucce e un modulo di memoria. Inserite il modulo di memoria nel vostro computer e memorizzate su di esso le istruzioni per il robot. Quando inserite il modulo nel robot, esso eseguirà quelle istruzioni alla lettera.

Capitolo

L'installazione del sistema MSX

I computer non escono dalle loro scatole già pronti per esser usati. Dovete effettuare alcune operazioni per farli funzionare. Fortunatamente, i computer MSX sono facili da montare: bastano solo pochi minuti e non serve alcun attrezzo.
Ogni computer MSX si monta in modo diverso, perciò le informazioni in questo capitolo sono generali e non specifiche di un particolare computer.

4.1 Montaggio

Fondamentalmente ci sono solo quattro fasi nel montaggio di un computer MSX:

- Estrarre il computer e tutti i componenti dalle scatole
- Leggere le istruzioni allegate al computer
- Collegare il computer alla televisione o al monitor
- Inserire la spina del computer e accenderlo

Tutte queste fasi sono molto semplici, ma la maggior parte delle persone trascura la più importante: leggere le istruzioni. Non si può fare a meno di mettere in evidenza l'importanza di questa fase. Dovreste leggere le istruzioni anche se le trovate terribilmente complicate o incredibilmente noiose.
Dopo aver tolto tutti i componenti dalle scatole, non gettate l'imballag-

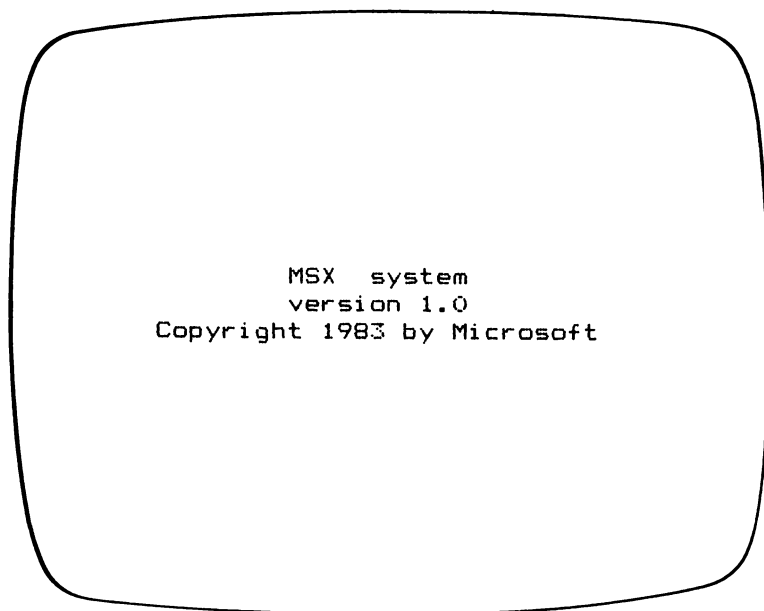



Figura 4.1 Schermo iniziale dell'MSX

gio: vi tornerà utile se dovrete trasportare il computer ed è necessario se dovete restituirlo. Conservate tutto il materiale stampato, incluso nella confezione, in un luogo preciso (un raccoglitore o una cartella vanno benissimo). Inoltre dovrete leggere la documentazione o almeno la parte riguardante il montaggio e la manutenzione del vostro computer; potete riservare ad un momento successivo la parte dedicata all'MSX-BASIC. A seconda della configurazione del computer MSX, potete trovare nella scatola dell'hardware in aggiunta all'unità principale. Alcuni computer MSX sono dotati delle periferiche citate nel Capitolo 3.
Se intendete usare la vostra televisione come video, il computer può essere dotato di un deviatore di antenna, che collega il computer al televisore. Innestate i cavetti alla presa dell'antenna sul retro del televisore e il computer nell'altra uscita del deviatore di antenna. Se userete invece un monitor dovreste trovare l'apposito cavo nella confezione. Una volta collegato il computer al monitor o al televisore (e dopo aver collegato tutti gli altri dispositivi hardware) siete quasi pronti per infilare la spina del vostro computer e accenderlo.
Seguite le istruzioni del costruttore per inserire la spina del vostro computer; potreste dover usare un trasformatore che ha una spina standard e una di forma differente. Alcuni computer lo hanno già all'interno ed hanno semplicemente una spina normale. Se usate un televisore seguite le istruzioni nel manuale del computer per il canale su cui sintonizzarvi.

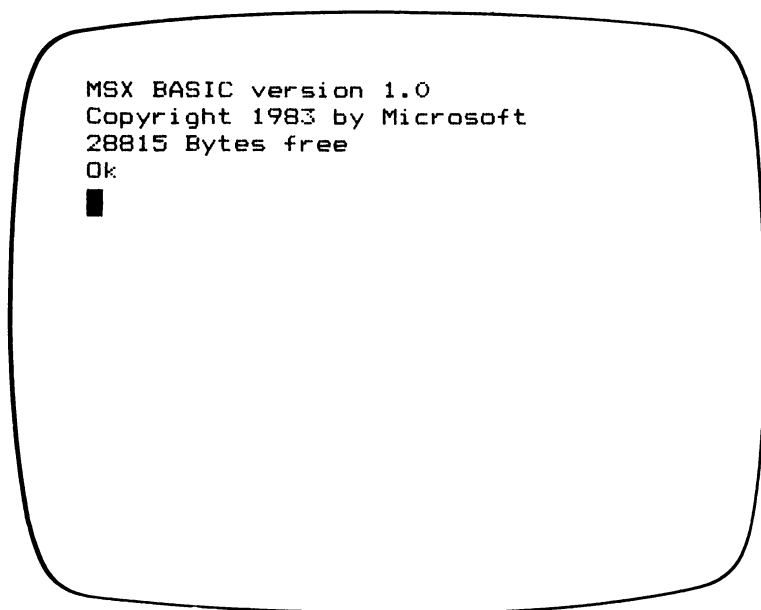



Figura 4.2 Schermo iniziale dell'MSX-BASIC

Infine, accendete il televisore (tenete abbassato al minimo o comunque molto basso l'audio), quindi accendete il computer. Se avete seguito correttamente le istruzioni, lo schermo mostrerà una scritta con il copyright MSX (Figura 4.1); poi vedrete la conformazione iniziale dello schermo dell'MSX-BASIC, simile a quella mostrata in Figura 4.2.
Se non vedete lo schermo di partenza dell'MSX-BASIC, non spaventatevi: il computer può essere programmato per far partire un programma diverso dall'MSX-BASIC. Se non vedete assolutamente nulla sullo schermo, può essere che non abbiate collegato correttamente il computer al monitor o al televisore.

4.2 Come usare le cartucce

Usare le cartucce MSX è generalmente molto facile, ma esistono alcune regole importanti circa la loro installazione. La regola più importante è di non inserire o estrarre una cartuccia dallo slot mentre il computer è acceso. Così facendo potreste danneggiare il programma sulla cassetta, ed è anche possibile che danneggiate il computer. Quindi, accertatevi che il computer sia spento prima di maneggiare la cartuccia. Siccome una cartuccia MSX è rettangolare, ci sono otto modi possibili per introdurla

nello slot, ma solamente uno è quello corretto, con la tacca in basso a sinistra. Solitamente c'è un'etichetta sulla cartuccia e deve essere rivolta verso di voi.

4.3 Come avere cura del sistema

Un computer MSX non richiede manutenzione come un'automobile. Comunque, ci sono poche semplici indicazioni da seguire che vi aiuteranno a far funzionare senza problemi il vostro sistema:

- Trattate con cura l'unità principale e le periferiche. Non mangiate o bevete mentre state usando il computer, poiché le briciole o il liquido accidentalmente rovesciato sulla tastiera provocano danni gravi, a volte difficili da scoprire.
- Tenete gli animali domestici lontani dal computer. La saliva e il pelo sono dannosi quanto le bibite e il cibo.
- Coprite la tastiera quando non la usate. State molto attenti se c'è polvere e polline nell'aria. Usate una copertura di tela o di materiale plastico antistatico.
- Non lavate il vostro computer; potete detergerlo con un panno appena umido, ma state attenti a non fare entrare acqua all'interno.

Per concludere, seguite tutte le altre precauzioni elencate nel manuale allegato al computer.

Parte Seconda

L'MSX-BASIC

Capitolo

Introduzione all'MSX-BASIC

Molte persone credono di dover sapere programmare per poter usare un computer. Mentre questo era vero vent'anni fa, non è più certamente il caso dei nostri giorni. I programmi disponibili in commercio possono svolgere praticamente qualsiasi funzione vogliate: come abbiamo visto, non avete bisogno di conoscere la programmazione per usare un programma di word processing.
Potreste pensare che, siccome il vostro computer MSX è dotato del linguaggio di programmazione MSX-BASIC, voi "dobbiate" imparare ad usarlo. State tranquilli: non avete alcun obbligo di imparare a programmare in MSX-BASIC. D'altra parte potrebbe interessarvi imparare a programmare in modo da poter conoscere meglio il computer.
Imparare a programmare è come imparare a riparare l'automobile: non è necessario ma a molte persone piace arrangiarsi. Spesso si acquisisce un nuovo rispetto e maggior confidenza per l'auto quando si impara a ripararla. Mentre imparare l'MSX-BASIC non aumenterà in modo significativo il vostro rispetto per i computer, potrete forse apprezzare l'intuito che acquisterete nel corso dell'apprendimento. La differenza tra riparare un'auto e programmare, è che nel primo caso risparmierete un sacco di soldi, mentre raramente risparmierete denaro scrivendo i vostri programmi. Inoltre, se conoscete la meccanica dell'automobile, potete discutere di miscele aria-benzina e di rapporto di compressione con gli amici, mentre se conoscete la programmazione dei computer, potete discutere di routine recursive e di puntatori indiretti.
Ci sono dozzine di linguaggi per computer che potete imparare, ma per molte ragioni l'MSX-BASIC rappresenta un buon linguaggio per cominciare; tra queste:

- **Convenienza.** Siccome l'MSX-BASIC è incluso nel vostro computer, non avete bisogno di comprare altro. Potete semplicemente accendere il vostro MSX ed usare l'MSX-BASIC.
- **Popolarità.** Quasi tutti i personal computer costruiti negli ultimi dieci anni usano una versione del BASIC ed esistono centinaia di libri su come programmare in BASIC.
- **Facilità di apprendimento.** Il BASIC è uno dei più semplici linguaggi per computer, e quindi uno dei più facili da imparare. Sebbene nell'MSX-BASIC ci siano più di 100 comandi e funzioni, i concetti che ne regolano l'uso sono facili da capire.

Molti esperti di computer disprezzano il BASIC perché i programmi scritti in questo linguaggio non sono eleganti come quelli scritti in altri linguaggi e affermano che se imparate il BASIC come vostro primo linguaggio per computer, non sarete più in grado di apprezzare la bellezza di alcuni altri linguaggi disponibili, più potenti e strutturati.

5.1 Cosa significa programmare

Lo scopo di scrivere un programma in qualsiasi linguaggio è quello di dare al computer una serie di istruzioni da eseguire. Per esempio, se volete che il vostro computer calcoli la somma di una serie di numeri e ne trovi la media, dovrete scrivere un programma che specifica ogni passo richiesto per ottenere tale scopo. Dare istruzioni al computer è lo stesso che dare istruzioni a una persona ragionevole: il computer può eseguire le istruzioni oppure dirvi che non capisce ciò che gli chiedete di fare.

DEFINIZIONE DI UN PROGRAMMA

Un programma è quindi semplicemente una sequenza di istruzioni scritte in un linguaggio che il computer può capire. A questa serie di istruzioni può essere dato un nome che la descriva. Per esempio, se dite a un bambino come portare fuori la spazzatura, dovete dare queste istruzioni: "Prendi il sacchetto che è sotto il lavandino e mettilo nel bidone. Se è lunedì sera, metti il bidone fuori sul marciapiede, altrimenti, assicurati semplicemente che il coperchio sia ben chiuso". La prossima volta che direte "porta fuori la spazzatura", il bambino, da questa più breve descrizione, saprà di dover eseguire la serie di istruzioni che avete spiegato precedentemente.
Programmare un computer è come insegnare a un bambino. Il seguente pseudo-programma spiega al computer come portare fuori la spazzatura:

```
Per portare fuori la spazzatura:
    1) Prendere il sacchetto da sotto il lavandino
       e metterlo nel bidone.
    2) Controllare se è lunedì.
          Se è lunedì mettere il bidone sul marciapiede,
          altrimenti controllare che il coperchio sia chiuso.
            Se non è chiuso, chiuderlo.
```

In questo esempio ci sono molte delle caratteristiche che si trovano in un tipico programma per computer.

- **Organizzazione.** I programmi dicono sempre al computer qual è l'ordine nel quale vengono eseguiti i compiti. Un computer comincia dall'inizio del programma e lo legge fino alla fine, eseguendo le istruzioni nell'ordine in cui le trova.
- **Comandi diretti.** Questi sono comandi incondizionati che il computer deve eseguire in ogni caso (come "prendere il sacchetto da sotto il lavandino").
- **Informazioni variabili.** In questo programma, l'informazione variabile è il giorno (lunedì) e la posizione del coperchio. Un altro esempio potrebbe prevedere che il computer debba mettere il bidone fuori sul marciapiede "a meno che sia vuoto"; in questo caso il computer dovrebbe anche controllare il peso del bidone.
- **Azioni condizionate.** Il computer prende decisioni ed esegue una particolare azione se una certa condizione è vera, ma esegue un'azione differente se la condizione è falsa. In questo caso la condizione è "oggi è lunedì": se così è, il computer porta fuori il bidone, altrimenti controlla il coperchio; i programmi in MSX-BASIC frequentemente svolgono azioni condizionate.

Come vedete, dire a un computer come svolgere un lavoro è come spiegarlo a una persona. Quando scrivete un programma, la cosa più importante da tenere a mente è di essere precisi nell'ordine dei comandi e delle azioni condizionate. Naturalmente, se questo fosse tutto quello che dovete ricordare, programmare sarebbe un gioco da ragazzi!

5.2 Comunicare con il computer

Scrivere un programma per computer è però parzialmente diverso che spiegare una sequenza di istruzioni a una persona perché dovete imparare un diverso linguaggio per comunicare e impartire comandi. Ciò può risultarvi difficile almeno per un po' di tempo.
Il linguaggio di programmazione è quello che usate quando dite al com-

puter cosa deve fare. Sfortunatamente non potete programmare in una lingua corrente perché i computer non ne capiscono abbastanza (anche se rimarrete stupiti da quanto inglese alcuni di essi capiscono). Dato che il computer MSX capisce l'MSX-BASIC, questo è il linguaggio che dovete imparare. Il vostro computer MSX non vi parla però in MSX-BASIC: solitamente, anche se non sempre, vi risponderà in inglese.
Uno degli aspetti più frustranti della programmazione è che si tratta prevalentemente di comunicare a senso unico: quello che voi fate in una eroica ora di battitura può produrre solo una misera risposta di cinque parole da parte del computer.
Normalmente il computer non vi risponde affatto a meno che non gli diciate di farlo. È come quando dite ad un bambino di portare fuori la "spazzatura": generalmente, la sola risposta che sentite è un occasionale "uffa". Se dite al computer come portare fuori la spazzatura, l'unico caso in cui vi dirà qualcosa sarà quando non capisce un'istruzione. Queste "cattive" istruzioni fanno apparire sullo schermo un messaggio di errore che nell'MSX-BASIC è scritto in inglese. I messaggi di errore vi aiutano ad individuare cosa c'è di sbagliato in ciò che avete scritto. Non pensate che sia impossibile conversare con il computer; potete farlo dando le istruzioni appropriate. Per esempio, le vostre istruzioni possono dire al computer come porvi le domande e come ribattere alle particolari risposte da voi date. Naturalmente dovrete dare queste istruzioni in un linguaggio di programmazione.
Per diventare bravi a programmare, dovete capire il linguaggio nel quale programmate: ogni linguaggio per computer ha un ben determinato vocabolario e ben definite regole di sintassi proprio come i linguaggi umani. Il vocabolario ha un numero limitato di verbi, nomi e aggettivi. Quando fate un errore nel dire al computer cosa fare, di solito è perché avete usato una parola che non è nel suo vocabolario o perché avete usato una costruzione sintattica errata. L'MSX-BASIC ha un vocabolario che è facile da imparare perché i verbi sono molto simili ai corrispondenti in inglese. Inoltre, ci sono solamente pochi nomi e aggettivi da imparare e la sintassi è molto lineare.

5.3 Conclusioni

Ora che ne sapete di più sulla programmazione, vi chiederete come mai c'è chi vuole fare la fatica di scrivere programmi. Alcune persone trovano piuttosto interessante lo stipendio da programmatore professionista. Altri si divertono semplicemente a programmare; lo considerano come un'estensione della propria mente.
Molte persone coltivano l'illusione di poter scrivere tutti i programmi di

cui hanno bisogno con il linguaggio incluso nel loro computer MSX: può essere una possibilità interessante, anche perché alcuni programmi disponibili in commercio sono piuttosto costosi. La triste verità è che per scrivere un programma complicato spesso occorrono settimane o mesi anche al miglior programmatore: la morale è che è di solito più conveniente comprare un programma, piuttosto che scriverlo.
Questo non vi deve scoraggiare dall'imparare a programmare. Come detto nel Capitolo 2, ci sono molte buone ragioni per imparare l'MSX-BASIC. Comunque, non dovete cominciare a imparare il linguaggio pensando che lo userete per scrivere tutti i programmi di cui possiate aver bisogno.
L'MSX-BASIC è un linguaggio adatto per scrivere programmi di gioco. Dato che esistono molti libri che contengono listati di programmi di gioco in MSX-BASIC, potete risparmiare una discreta quantità di denaro ricopiando questi programmi invece di comprare le cartucce. Ribattendo un programma potete farvi anche un'idea del modo in cui il programma è stato sviluppato.
Dopo aver giocato, potete lievemente modificare il programma per adattarlo ai vostri particolari interessi o capacità. Ricordate che la ragione più importante di scrivere programmi è imparare di più sul funzionamento del vostro MSX e dei computer in generale. Quando scriverete programmi e li proverete, vi renderete conto di avere una migliore predisposizione per la logica e per l'ordine che il computer richiede. Capirete meglio anche i limiti del computer e le particolari limitazioni dell'MSX-BASIC.
Dopo aver imparato l'MSX-BASIC e scritto un po' di programmi, forse vorrete imparare un nuovo linguaggio. Buona parte dei linguaggi hanno molto in comune con il BASIC, quindi impararne uno nuovo sarà più facile di quanto non sia stato imparare il primo.
A parte la scelta del linguaggio, scrivere programmi vi aiuterà a saperne di più sul vostro computer. Se approfondite la vostra conoscenza della programmazione, potreste anche scoprire idee nuove per organizzare dati e procedure. In questo modo imparare a programmare vi aiuterà a capire meglio le informazioni e la logica.

Capitolo

Il primo programma 6

Il modo più semplice per imparare l'MSX-BASIC è quello di ricopiare un programma e analizzarlo. Questo capitolo vi mostra come battere i programmi, salvarli su nastro o su disco e farli girare. Imparerete anche alcune tecniche da usare nella stesura e nella modifica dei programmi. Il Capitolo 7 parlerà delle differenti parti di un programma. I Capitoli dall'8 al 16 vi diranno di più sugli specifici comandi e funzioni dell'MSX-BASIC.
Il vostro computer è certamente dotato di un manuale di riferimento dell'MSX-BASIC. Questo manuale, più che insegnare il linguaggio ne descrive le varie parti. Quando leggete questi capitoli, dovrete consultare il manuale per maggiori dettagli sui comandi. Poiché questo libro si occupa nel complesso dei sistemi MSX, non può essere dettagliato come il vostro manuale. Usate questo libro come aiuto per l'apprendimento e il manuale come testo di riferimento.

6.1 Avviare l'MSX-BASIC

Per poter programmare in MSX-BASIC, dovete caricare il linguaggio nel computer. Se accendete il computer senza nessuna cartuccia nel connettore, esso farà partire automaticamente l'MSX-BASIC. Attualmente, quasi tutti i computer MSX funzionano in questo modo. Se il vostro non lo fa consultate il manuale per capire come caricare l'MSX-BASIC. Se avete un drive con la sua cartuccia inserita nel connettore, per far partire l'MSX-BASIC seguite le istruzioni date all'inizio del Capitolo 18.

Se avete il drive, avrete anche un programma chiamato MSX-DISK BASIC che è una versione ampliata dell'MSX-BASIC, il che significa che funziona più o meno allo stesso modo avendo però a disposizione alcune caratteristiche in più. Quando fate partire l'MSX-BASIC con il drive, in realtà fate partire l'MSX-DISK BASIC. Tutte le caratteristiche in più sono relative all'uso dei dischetti; chi non ha un drive non perde nulla per il fatto di non avere l'MSX-DISK BASIC. I comandi dell'MSX-DISK BASIC vengono citati in questo libro assieme a quelli dell'MSX-BASIC: se non avete un drive, ignorateli tranquillamente.

6.2 L'uso dei comandi MSX-BASIC

Una volta partito l'MSX-BASIC, se avete il drive inserito, sul vostro schermo apparirà la videata di Figura 6.1. "Ok" è un messaggio dell'MSX-BASIC, che significa che è pronto ad accettare le vostre istruzioni. C'è inoltre una linea di parole nella parte bassa dello schermo, che esamineremo più avanti. Il rettangolino che appare sotto il messaggio Ok è chiamato cursore. Il cursore è importante in MSX-BASIC, poiché indica dove apparirà la prossima lettera che batterete. Per verificare ciò, battete qualche lettera e osservate che esse appaiono dove si trova il cursore e

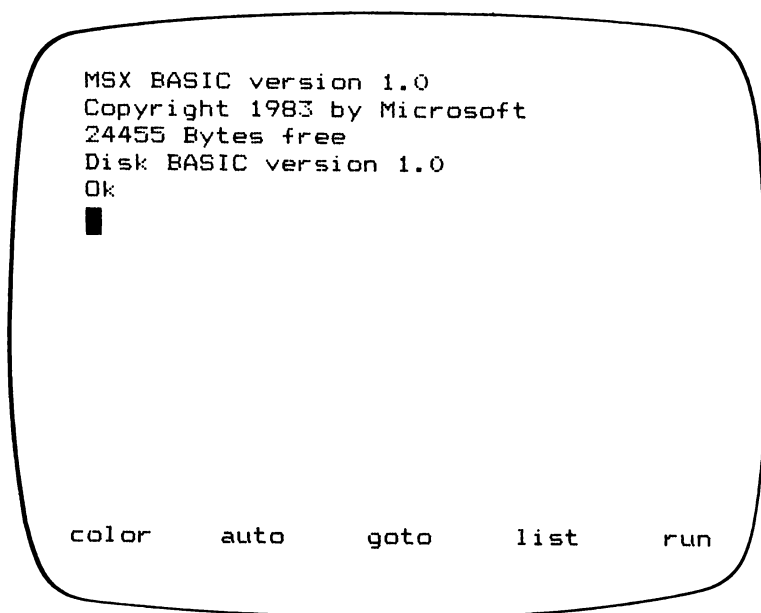



Figura 6.1 Videata iniziale dell'MSX-DISK BASIC

che il cursore si muove verso destra ogni volta che battete una lettera. Per cancellare ciò che avete scritto, premete il tasto denominato BS o BACKSPACE. Il tasto BACKSPACE funziona proprio come quello di ritorno della macchina da scrivere, tranne che quando muovete il cursore a sinistra, esso cancella i caratteri.
Siete pronti a dare il primo comando in MSX-BASIC. Accertatevi che il cursore sia sul margine sinistro (altrimenti usate il tasto BACKSPACE) e battete:

```
LIST
```

(Non importa se scrivete in lettere maiuscole o in lettere minuscole o in entrambi i modi; in questo libro useremo sempre le maiuscole.)
Poi premete il tasto RETURN; questo può essere designato da una freccia che punta in basso e a sinistra o dalla parola RETURN.
Premere il tasto RETURN significa che volete che l'MSX-BASIC esegua il comando che avete scritto su quella linea. In questo caso volete che l'MSX-BASIC esegua il comando LIST.
Quando premete il tasto RETURN, l'MSX-BASIC vi risponde Ok. Questa, decisamente, non è una risposta molto emozionante, specialmente perché l'MSX-BASIC vi dice Ok anche quando non state facendo nulla. Comunque, visto che l'MSX-BASIC non ha segnalato un errore, significa che avete dato un comando che ha riconosciuto. Se l'MSX-BASIC avesse stampato il messaggio di errore

```
Syntax error
```

prima di stampare Ok, questo avrebbe voluto dire che non ha capito il comando che gli avete dato. Nel nostro esempio, se ottenete un messaggio di errore, dovete aver scritto LIST in modo scorretto. Per puro divertimento provate a provocare un errore. Battete

```
LISR
```

e premete RETURN. Vedrete

```
Syntax error
Ok
```

Non importa quello che fate, il calcolatore vi dirà sempre Ok quando è pronto per ricevere i vostri comandi. "Syntax error" è il modo che l'MSX-BASIC usa per dire che avete dato un comando che non conosce,

cioè che non si trova nel suo vocabolario. Come potete vedere, l'MSX-BASIC riconosce il comando LIST ma non la parola LISR.

CORREGGERE GLI ERRORI DI BATTITURA

L'MSX-BASIC vi permette di posizionarvi in un qualunque punto dello schermo usando i tasti di controllo del cursore. Come ricordate, i tasti di controllo del cursore sono quelli con sopra le frecce, solitamente posizionati alla destra della tastiera.
Premete tre volte il tasto ⇑ ; notate che il cursore si è mosso verso l'alto e si trova ora sulla L di LISR. Premete tre volte il tasto ⇒ e il cursore è ora sulla R di LISR (potete probabilmente immaginare cosa succederebbe premendo i tasti ⇐ e ⇓). Poiché volete cambiare LISR in LIST dovete sostituire la R con una T. L'MSX-BASIC rende questa operazione molto facile: dovete semplicemente battere una T sopra la R. Notate che ora il vostro comando è LIST. Battere un comando sopra un altro significa sostituirlo del tutto: la R è stata dimenticata, come se non l'aveste mai battuta; l'operazione è chiamata *battitura sovrapposta*. Quando premete RETURN lo schermo apparirà come in Figura 6.2. "Okntax error", ovviamente, sembra un po' strano. Ma quando pensate a ciò che ha fatto il computer, tutto ciò acquista significato. Quando avete cambiato LISR in LIST,

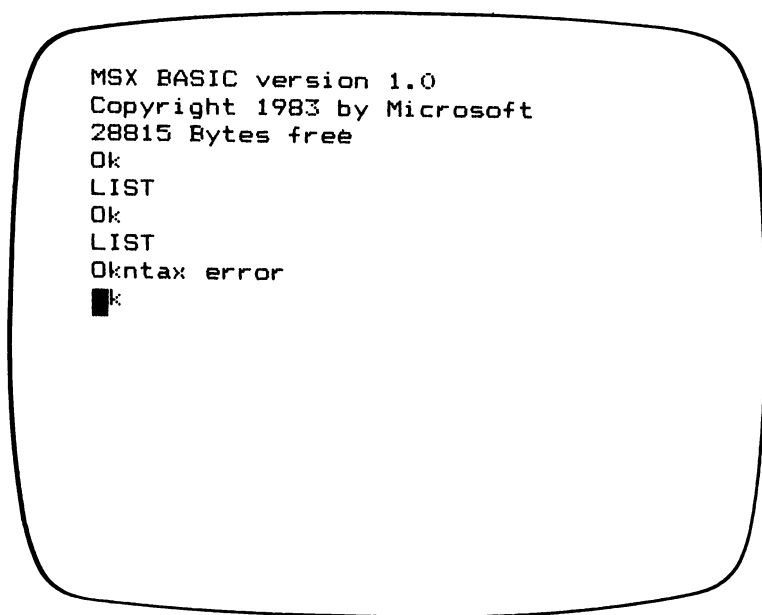



Figura 6.2 Effetto del secondo comando LIST

che è un comando valido, esso ha risposto esattamente come aveva fatto prima: stampando Ok sulla linea successiva e piazzando il cursore sulla linea dopo l'Ok. Ha ignorato del tutto ciò che si trovava sullo schermo prima della correzione. L'Ok su cui si trova il cursore è rimasto sullo schermo da quando avete dato il comando sbagliato LISR.
L'MSX-BASIC vi permette di correggere gli errori in molti modi; ovviamente la battitura sovrapposta è il più semplice. Per vedere gli altri, battete sull'Ok che si trova sopra la linea del cursore, la parola LIRST, e premete RETURN. Ancora una volta ottenete l'onnipresente messaggio "Syntax error". Usate i tasti di controllo del cursore per posizionarlo sulla R di LIRST e premete il tasto con la scritta DEL o DELETE. Notate che la lettera che si trova sotto il cursore (la R) scompare quando usate il tasto DEL. Quando premete il tasto RETURN, il computer esegue il comando LIST.
Un altro tasto utile da conoscere è il tasto INS, talvolta indicato sulla tastiera con INSERT. Per vedere come funziona il tasto INS, date il comando LT (che provocherebbe un errore). Ora muovete il cursore sulla T di LT e premete il tasto INS. Noterete che il cursore si restringe ad una sottolineatura della lettera. Ora battete I, e notate come la lettera viene inserita a sinistra del cursore, mentre la lettera sotto il cursore con tutte quelle alla sua destra si spostano per far spazio alla nuova lettera. Battete S e premete RETURN.
La lezione da imparare da tutto questo è che l'MSX-BASIC vi permette di correggere gli errori di battitura, anche dopo che avete premuto RETURN. Dovete semplicemente far tornare il cursore alla linea che avete sbagliato e correggerla con una ribattitura, cancellando o inserendo caratteri. Quando premete RETURN, il calcolatore accetta la nuova linea e dimentica la precedente.

6.3 Come scrivere un semplice programma

Ora che sapete come battere i comandi in MSX-BASIC e come correggere gli eventuali errori che potete commettere, siete pronti a scrivere il vostro primo programma. Naturalmente, finché non conoscerete meglio la programmazione, non potrete scrivere programmi complessi: non preoccupatevi troppo di questo, perché molto presto scoprirete altre cose sui programmi.
Quello che segue è un tipico programma in MSX-BASIC. Notate che ogni linea incomincia con un numero e che i numeri crescono lungo il programma. L'MSX-BASIC usa questi numeri per determinare l'ordine in cui fare le cose.
Scrivete il programma con molta attenzione e controllate ogni linea dopo

averla scritta. L'MSX-BASIC non vi dirà se ci sono errori di sintassi se non più avanti, è quindi meglio evitarli fin da ora con una attenta trascrizione. Se vi accorgete di avere fatto un errore, correggetelo usando i metodi imparati nel precedente paragrafo.

```
10 REM Il mio primo programma in MSX-BASIC
20 CLS
30 PRINT "Ecco un po' di grafica ..."
40 REM Inizializza il tempo, e attende 2 secondi
50 TIME = 0
60 IF TIME < 100 THEN GOTO 60
70 REM Entra in modo grafico
80 SCREEN 2
90 REM Disegna alcuni cerchi concentrici
100 FOR I = 10 TO 50 STEP 5
110 CIRCLE (126,86), I
120 NEXT I
130 REM Aspetta ancora 2 secondi
140 TIME = 0
150 IF TIME < 100 THEN GOTO 150
160 REM Disegna casualmente dei rettangoli
170 FOR J = 1 TO 50
180 LINE (RND(1)*255,RND(1)*195) - STEP(RND(1)*50,
RND(1)*50),RND(1)*15,BF
190 NEXT J
200 REM Aspetta ancora 2 secondi
220 IF TIME < 100 THEN GOTO 220
230 SCREEN 0
240 PRINT "Tutto fatto"
```

Battendo una linea particolarmente lunga, come la 180, quando arrivate al bordo dello schermo, continuate a scrivere (non premete RETURN) e l'MSX-BASIC proseguirà sulla successiva linea dello schermo. Tutte le linee di listato che superano le quaranta colonne (la larghezza del vostro schermo) devono continuare sulle linee seguenti. Comunque, quando scrivete in più linee, fatelo sempre come se si trattasse di una sola lunga linea; altrimenti l'MSX-BASIC non capirà e vi darà un messaggio di errore.

6.4 Come far girare il programma

Scrivendo il vostro primo programma, può darsi che abbiate trovato parti gia comprensibili. Poiché alcune delle linee che avete scritto si sono spostate fuori dallo schermo (questo è anche conosciuto come *scrolling* verso l'alto dello schermo), per rivedere tutto il vostro programma dovete dare un comando LIST: ciò è anche consigliabile per controllare se ci so-

no errori di battitura nel programma. Comunque, siete probabilmente più interessati a far girare il programma per vedere cosa fa. È quindi il momento di imparare il vostro secondo comando MSX-BASIC: RUN. Battete RUN e premete RETURN. Come avete visto dovete premere RETURN dopo tutti i comandi: d'ora in poi, quando vi sarà detto di dare un comando, dovrete sempre premere RETURN dopo di esso. Ecco cosa succede, in sequenza, se non avete commesso nessun errore nella battitura del programma:

1. Lo schermo si ripulisce.
2. Appare per due secondi il messaggio "Ecco un po' di grafica ...".
3. Lo schermo si ripulisce nuovamente e vengono disegnati sullo schermo otto cerchi concentrici.
4. Due secondi circa più tardi, appaiono cinquanta rettangoli di varie dimensioni e vario colore, sparsi per lo schermo.
5. Dopo due secondi tutti i rettangoli sono sullo schermo, lo schermo si ripulisce di nuovo e vedrete il messaggio "Tutto fatto" e il solito Ok.

Se avete sbagliato a scrivere qualcosa l'MSX-BASIC individua l'errore, pulisce lo schermo e stampa il messaggio di errore sulla linea più in alto, e il messaggio Ok sulla linea seguente. L'MSX-BASIC blocca l'esecuzione del programma appena trovato un errore. Non preoccupatevi di questo per ora; imparerete a riconoscere gli errori nel Capitolo 7.

6.5 Come rivedere il programma

Per ora supponiamo che il vostro programma abbia funzionato a dovere e che quando lo avete fatto girare abbiate visto quello che è stato appena descritto. Ora lo schermo è pulito e non potete vedere nulla del programma che avete scritto. Per far listare l'intero programma dal computer battete LIST e premete RETURN; ora il listato del programma appare sullo schermo. Notate che parte del programma è finito fuori dallo schermo, perché troppo lungo. Nel Capitolo 7 imparerete come listare solo poche linee di programma.

Se avete una stampante collegata al vostro MSX potete usare il comando LLIST per far stampare il listato su carta anziché sul video. Accertatevi che la stampante sia collegata correttamente, che sia accesa e che la carta sia pronta. Date il comando LLIST e premete RETURN.

6.6 Come salvare il programma su nastro o su disco

Ricordate dal Capitolo 1 che quando spegnete il computer vanno perdute tutte le informazioni contenute nella RAM, che è il luogo dove il BASIC ha memorizzato il programma che avete scritto. Per evitare di doverlo riscrivere, dovete salvarlo su nastro o su disco. Fortunatamente, questo è abbastanza semplice. Anche se il programma contiene degli errori che non avete potuto individuare, potete ugualmente memorizzarlo.
Se avete un mangianastri, seguite le istruzioni del manuale del vostro computer, per collegarlo. Se avete un drive seguite le istruzioni sul manuale del drive per collegarlo al sistema. Ora vedremo come preparare il registratore o il drive per salvare il programma e i comandi necessari per salvare il programma e per caricarlo nella RAM dopo averlo salvato.

USO DEL MANGIANASTRI

L'MSX-BASIC distingue due tipi di mangianastri: quelli con comando a distanza del motore e quelli senza. Se ne avete uno con comando a distanza, il computer lo potrà controllare al vostro posto. Diversamente sarete voi a premere il tasto PLAY sul registratore ogni volta che volete far scorrere il nastro.
Riavvolgete il nastro fino all'inizio. Se il vostro registratore ha il comando a distanza, quando premete il tasto di riavvolgimento non succede nulla; dovete staccare prima il cavo inserito nella presa del controllo a distanza.
L'MSX-BASIC immagazzina i programmi sulla cassetta nello stesso modo in cui una piastra di registrazione stereo registra la musica e cioè dall'inizio alla fine del nastro. Se con il vostro stereo registrate la canzone "Bianco Natale" all'inizio della cassetta, riavvolgete e registrate "Sapore di sale", la registrazione di "Bianco Natale" viene cancellata; nello stesso modo, se salvate un programma all'inizio della cassetta, riavvolgete e salvate un altro programma, il primo programma viene perso per sempre.
Quindi dovete stare attenti a dove registrate sulla cassetta. Avete due possibilità: registrare un solo programma per ogni lato della cassetta, o usare il contatore sul mangianastri per determinare dove è possibile registrare un nuovo programma. Usare un lato della cassetta per registrare un solo programma è il metodo più sicuro, ma comporta anche uno spreco di spazio, soprattutto quando il programma occupa solo pochi secondi di nastro. Ricopiarsi il numero del contatore per ogni programma salvato e posizionarsi sulle aree con l'avanzamento rapido è molto più economi-

co, ma aumenta anche la probabilità di perdere un programma. Ricordate che i contatori dei mangianastri più economici non sono molto precisi. A parte il metodo che usate dovete porre il registratore in fase di registrazione prima di salvare un programma su nastro (di solito premendo i tasti RECORD e PLAY contemporaneamente). Per caricare il programma nella RAM dovete premere il tasto PLAY.

USO DEL DRIVE

Dopo essere stato collegato nel connettore per cartucce, il drive è pronto per salvare e ricaricare programmi. Avete bisogno ovviamente di un dischetto da poter usare a questo scopo; in nessun caso dovete usare il dischetto dell'MSX-DOS per memorizzare programmi, ma dovete prepararne un altro. L'operazione di preparare i dischetti per la memorizzazione di programmi, è detta *formattazione* ed è trattata nel Capitolo 18. Se non avete familiarità con i concetti di formattazione di un disco, dovrete leggere quel capitolo prima di proseguire.
Ponete un dischetto vuoto nel drive. Ricordate che la formattazione di un disco cancella tutto ciò che si trova su di esso; perciò dovete stare particolarmente attenti a non formattare mai il dischetto dell'MSX-DOS allegato al drive.
Potete formattare un disco mentre sta girando l'MSX-DOS o l'MSX-BASIC. Per fare ciò date il comando CALL FORMAT. Poiché ogni fabbrica ha una maniera diversa di formattare i dischetti, possono apparire sullo schermo diverse scritte o opzioni, che troverete spiegate nel manuale del computer o in quello del drive. In caso contrario, leggete le opzioni e scegliete le più appropriate. Dopo essere stato formattato, il dischetto è pronto per immagazzinare programmi. Una buona idea è quella di mettere sul dischetto un'etichetta che ne indichi il contenuto (per esempio "Primi programmi in MSX-BASIC").

MEMORIZZARE IL PROGRAMMA

L'MSX-BASIC può memorizzare ciò che avete scritto se avete seguito le precedenti istruzioni per l'uso del drive o del mangianastri. Quando siete pronti per memorizzare il programma, date uno dei seguenti comandi:

Per la cassetta:	Per il dischetto:
`CSAVE "PRIMO"`	`SAVE "PRIMO"`

Quando date il comando il nastro inizierà a muoversi o il drive a fare ru-

more. Come potete immaginare, la C in CSAVE sta per cassetta. Quello che segue il comando è il nome del vostro programma. Quando volete recuperarlo, dovrete usare lo stesso nome. Ciò vi permette di salvare più di un programma sulla stessa cassetta o sullo stesso disco, poiché ogni programma avrà un nome diverso. Il nome può essere di sei o meno caratteri per il mangianastri oppure otto o meno per il drive, senza contare le virgolette. Nel nostro caso PRIMO ha cinque caratteri. Nel nome di un programma potete usare tutte le lettere maiuscole.
Ora che il vostro programma è al sicuro, potete spegnere il computer senza problemi.

CARICARE IN MEMORIA IL PROGRAMMA

Riaccendete il computer e date il comando LIST per accertarvi che non ci sia nessun programma in memoria. Se state usando un mangianastri, riavvolgete il nastro e premete PLAY. Poi battete uno dei seguenti comandi:

Per la cassetta:	Per il dischetto:
`CLOAD "PRIMO"`	`LOAD "PRIMO"`

Notate la relazione tra CLOAD e CSAVE e tra LOAD e SAVE. Date il comando LIST, per vedere se il programma è stato ricaricato.
Quando scriverete altri programmi ricordate di dare ad ognuno di essi un nome diverso quando lo salvate. A questo punto, tutto quello che avete fatto è scrivere un programma, che non è detto abbiate necessariamente capito, listarlo e salvarlo su nastro o su disco.
Il prossimo capitolo parla molto più diffusamente di cosa è un programma MSX-BASIC e prosegue nella descrizione del linguaggio.

Capitolo

La struttura dei programmi

Ora che avete scritto e salvato un programma sarete probabilmente curiosi di sapere che cosa in realtà sia e perché faccia le cose che avete visto sullo schermo quando lo avete fatto girare. Per capire questo dovete sapere qualcosa di più sulla struttura dell'MSX-BASIC. Il programma PRIMO è un buon esempio perché contiene molti elementi dell'MSX-BASIC.
Ricordate dal Capitolo 5 che ci sono due parti fondamentali in un linguaggio di programmazione: la sintassi e il vocabolario. Voi avete già imparato una piccola serie di vocaboli come LIST, RUN e SAVE. Avete anche imparato una discreta quantità di regole sintattiche. Come in molte lingue, la sintassi è molto più complicata del vocabolario. Le regole di sintassi che avete imparato sinora sono:

- Tutte le istruzioni vengono date scrivendo una parola chiave del vocabolario dell'MSX-BASIC e premendo RETURN.
- Quando scrivete un'istruzione, solitamente succede qualcosa in risposta ad essa.
- Un programma è una serie di istruzioni. Quando scrivete un programma, l'MSX-BASIC non reagisce immediatamente a quello che avete scritto.
- Per fare eseguire il programma al computer, si dà l'istruzione RUN.
- Alcune istruzioni si scrivono isolate su una riga (come RUN), mentre altre richiedono dopo di loro altre istruzioni (come SAVE e LOAD).
- Potete decidere i nomi per i programmi.

7.1 Come correggere le linee di programma

A questo punto, una buona idea sarebbe quella di controllare che il vostro programma PRIMO sia corretto. Se ha girato al primo tentativo, congratulazioni. In caso contrario, dovete determinare dove è l'errore. Fortunatamente, l'MSX-BASIC rende questa operazione estremamente facile. Fate eseguire il programma finché non ottenete il messaggio di errore. Notate che ora il messaggio ha qualcosa che il nostro vecchio "Syntax error" non aveva: un numero. Per esempio, se avete accidentalmente sbagliato a scrivere la linea 80 battendo:

```
80 SCREEM 2
```

dovreste ricevere il messaggio di errore:

```
Syntax error in 80
```

Quindi, "in 80" vi dice che l'errore di sintassi si trova alla linea 80. Potete ora correggere la linea 80 usando i comandi per la correzione, che avete imparato nell'ultimo capitolo.
L'MSX-BASIC può individuare molti errori ma non tutti. Per esempio, cambiate la linea 60 in questo modo:

```
60 IF TIME < 10 THEN GOTO 60
```

Questo è, chiaramente, diverso da ciò che intendevate scrivere. Comunque quando fate partire il programma, l'MSX-BASIC non può rilevare l'errore; il programma attenderà due decimi di secondo anziché i due secondi previsti, ma proseguirà correttamente.
Anche se i messaggi di errore dell'MSX-BASIC contengono un numero di linea, non è detto che quella sia la linea dove avete scritto qualcosa di errato; è la linea dove il calcolatore è rimasto confuso. Quando programmerete di più, otterrete spesso dei messaggi di errore che indicano qualcosa di sbagliato in una linea che sembra perfetta. In questo caso, guardate le linee immediatamente precedenti a quella indicata per vedere se qualcos'altro può avere causato il problema.
Se scoprite di aver fatto un errore in una linea prima della 50, noterete un altro problema: ogni volta che date il comando LIST, quelle linee escono fuori dallo schermo e non potete raggiungerle per correggerle. C'è una facile soluzione: usate il comando LIST con il numero della linea. Per esempio, se avete scritto

```
20 CLD
```

e volete cambiare la linea, date il comando:

```
LIST 20
```

Ora solo la linea 20 viene listata, così è semplice correggerla. Potete dare il comando LIST anche con un insieme di numeri. Per esempio, se volete vedere solo le linee dalla 20 all'80, date il comando:

```
LIST 20-80
```

Rivedete il vostro programma in modo che quando lo farete girare, agisca come previsto.

7.2 Le parti di una linea

Come avete visto, un programma in MSX-BASIC è costituito da linee. Il modo più semplice per esaminare la sintassi dell'MSX-BASIC è capire prima la sintassi di una sola linea. Fatto questo, il resto della sintassi (e buona parte del vocabolario) è abbastanza facile.
Ci sono tre parti in ogni linea di un programma:

- Il numero di linea
- Il comando, l'istruzione o la funzione
- Le informazioni pertinenti al comando, l'istruzione o la funzione

L'ordine degli elementi nella linea è fisso: il numero viene per primo, il comando viene immediatamente dopo il numero di linea (con uno spazio tra i due) e le altre informazioni seguono il comando.
L'MSX-BASIC ha alcune restrizioni per quanto riguarda le linee. Una linea deve avere meno di 255 caratteri, ma questo non costituisce un problema a meno che abbiate sequenze molto lunghe o vogliate mettere molti comandi su una stessa linea. Inoltre, il numero di linea deve essere compreso tra 1 e 65529. Le parole o i numeri che vengono dopo un comando, un'istruzione o una funzione sono detti argomenti. Molti comandi hanno argomenti, ma non tutti. Potete pensare all'MSX-BASIC come ad una lingua, con i comandi al posto dei verbi e gli argomenti al posto dei nomi o avverbi. Esaminando il programma, potete probabilmente immaginare altre regole di sintassi:

- Alcuni comandi possono avere più di un argomento
- Alcuni argomenti sono numeri, altri sono testo e altri sono misti

Potete anche aggiungere una quantità di nuovi termini al vostro vocabo-

lario: REM, CLS, PRINT, IF THEN, SCREEN, FOR, CIRCLE, NEXT e LINE. Alcuni di questi saranno studiati in questo capitolo; vedremo gli altri nei capitoli successivi.
Il lettore attento avrà notato che nella lista precedente non compare la parola TIME che appare nelle linee 50, 140 e 210. TIME non fa parte del vocabolario dell'MSX-BASIC. In queste linee è stata usata un'abbreviazione utilizzata da tutti i programmatori BASIC: l'istruzione LET è stata tralasciata. Quindi la linea 50 avrebbe potuto essere scritta:

```
50 LET TIME = 0
```

LET è la sola parola riservata che potete tralasciare.
L'MSX-BASIC vi permette di mettere più istruzioni sulla stessa linea se sono separate con i due punti e se la prima non è una REM.
Per esempio le due linee

```
230 PRINT "Tigrotti"
240 PRINT "della Malesia"
```

possono essere scritte in una sola

```
230 PRINT "Tigrotti": PRINT "della Malesia"
```

Notate che non si usa un secondo numero di linea: entrambe le istruzioni sono considerate parte di una unica linea. Mettere più di un'istruzione su una linea però rende meno leggibile il programma. Comunque questo metodo è utile per IF THEN, come vedrete nel Capitolo 9.
Ora che conoscete le parti di una linea, vi sarà utile capire le relazioni tra le istruzioni e i loro argomenti. La Figura 7.1 elenca le istruzioni MSX-BASIC che abbiamo utilizzato finora, con l'indicazione del numero di argomenti dai quali queste possono essere seguite. Nella Figura 7.1, REM e LET hanno un solo argomento, anche se, guardando il programma, si potrebbe pensare che entrambe ne possano avere molti. Nell'istruzione REM, tutto quello che viene dopo la parola chiave viene ignorato, perciò costituisce un solo argomento, a prescindere dal numero di parole. Ogni LET è seguita da un'equazione del tipo "qualcosa uguale a qualcos'altro", come TIME=0, cosicché tutto questo è considerato un argomento. IF e FOR sono escluse dall'elenco poiché essendo istruzioni costituite da più parti, sono difficili da classificare.
Notate che le istruzioni PRINT nel programma hanno ognuna un solo argomento: tutto quello che si trova all'interno delle virgolette è un unico argomento. Le virgolette sono usate per definire l'argomento, indicando all'MSX-BASIC dove esso incomincia e dove finisce. Esse delimitano anche il nome del programma che è l'argomento dei comandi CSAVE, SAVE, CLOAD e LOAD.

Istruzioni	Senza argomenti	Un argomento	Più argomenti
LIST	□	□	□
RUN	□	□	□
CSAVE		□	□
SAVE		□	□
NEW	□		
CLOAD		□	□
LOAD		□	□
REM	□	□	
CLS	□		
PRINT	□	□	□
SCREEN		□	□
CIRCLE			□
NEXT		□	
LINE			□
LET		□	

Figura 7.1 Istruzioni e argomenti

Come vedete, è grande la quantità di comandi e istruzioni che possono avere più di un argomento. Scoprirete che la maggior parte può avere molti argomenti che svolgono differenti compiti.

7.3 Esempi di istruzioni MSX-BASIC

Ora che conoscete le parti di una linea di programma, potete iniziare a studiare le istruzioni che appaiono nel vostro primo programma. Quelle che imparerete in questo capitolo, appaiono in quasi tutti i programmi ed è quindi utile esaminarle ora. Alcune di queste istruzioni saranno trattate più dettagliatamente nei prossimi capitoli.
La prima linea del programma contiene una REM. REM è l'abbreviazione di *remark* (commento), che indica appunto la natura di questa istruzione. Le REM si inseriscono lungo il programma, per ricordare quello che fanno le varie parti del programma. Per esempio, nel programma PRIMO la linea 10 dichiara lo scopo di tutto il programma, la linea 40 vi dice quello

che fanno le linee 50 e 60, e la linea 70 vi spiega la 80 e così via.
Vi potreste chiedere perché, quando scrivete un programma, dobbiate mettere tante annotazioni su quello che intendete fare. Come i vostri programmi diventeranno più complicati, vi accorgerete di usare sempre più "trucchi" di programmazione; più lungo sarà il programma, più difficile sarà ricordare quale particolare compito svolge una certa parte o quale espediente avete usato in un certo punto. Sarà allora che apprezzerete le note lasciate con le REM.
Se volete modificare un programma o scoprire qualcosa che non funziona, potete perdere un sacco di tempo cercando lungo il programma l'area da cambiare. Se avete usato molte REM, che spiegano la funzione di ogni parte e spiegano sinteticamente i vari artifizi, sarete in grado di leggere il vostro programma molto più facilmente.
Non è difficile immaginare la funzione dell'istruzione PRINT guardando girare il programma d'esempio. PRINT stampa il suo argomento sullo schermo. Come avete imparato prima, l'argomento dell'istruzione PRINT è delimitato dalle virgolette. La linea 30 contiene un tipico PRINT; tutto quello che si trova tra le virgolette (ma non le virgolette stesse) viene stampato sul video.
Come detto prima, la linea 50 sottintende un LET, che viene soppresso in quasi tutti i programmi BASIC. L'argomento dopo il numero di linea consiste nel nome della variabile, in un segno di uguale e in un valore. Le variabili assumono i valori dei numeri, proprio come nell'algebra, e sono descritte più dettagliatamente nel Capitolo 8. Per ora, il concetto importante da capire è che LET modifica il valore delle variabili.

7.4 Organizzazione di un programma

Come scoprirete presto, l'MSX-BASIC esegue le linee di un programma dal numero di linea più basso al più alto. Volendo far eseguire all'MSX-BASIC certe istruzioni prima di altre, assume grande importanza il numero che viene assegnato ad ogni linea.
Una convenzione standard tra i programmatori BASIC è numerare le linee del programma di 10 in 10 anziché con incremento di 1. La ragione principale è che potreste avere bisogno più avanti di aggiungere dei comandi nel mezzo del vostro programma ed è perciò utile lasciare dello spazio a questo scopo. Per esempio, se volete inserire una linea tra la 20 e la 30, potete darle il numero 25. Potete scrivere la linea 25 in una qualsiasi riga libera dello schermo; dando il comando LIST la vedrete tra la linea 20 e la linea 30.
Buona parte dei programmi, comunque, non viene eseguita semplicemente dall'inizio alla fine, linea dopo linea. È tipico dei programmi "saltare"

avanti e indietro, il che significa che alcune linee vengono scavalcate in base a certe condizioni. Per esempio, potreste scrivere un programma che vi pone una domanda alla quale dovete rispondere sì o no. Se la risposta è sì, volete che venga eseguita una particolare sequenza di linee di programma, ma se la risposta è no, volete che ne venga eseguita un'altra. Per prendere queste decisioni usate le istruzioni IF e GOTO. Esse vengono spiegate più dettagliatamente nel Capitolo 9, ma vi sarà utile vederne un esempio fin da ora. Cancellate il programma che avete in memoria dando il comando NEW. Scrivete il breve programma che segue:

```
10 REM Esempio dell'uso di GOTO
20 LINE INPUT "Batti S o N: "; RISP$
30 IF RISP$ = "S" THEN GOTO 70
40 IF RISP$ = "N" THEN GOTO 90
50 PRINT "Non hai battuto S o N"
60 GOTO 100
70 PRINT "Hai battuto S"
80 GOTO 100
90 PRINT "Hai battuto N"
100 END
```

Potete salvare questo programma su nastro o su disco con il nome "SI O NO".
Fate girare il programma e battete una S maiuscola quando esso vi chiede "Batti S o N"; poi premete RETURN (accertatevi di aver battuto una S maiuscola e non minuscola). Il programma vi confermerà che avete battuto una S.
Sapete già quello che fa la linea 10: nulla. L'istruzione LINE INPUT alla linea 20, stampa il messaggio racchiuso tra virgolette e poi attende che scriviate alcuni caratteri e premiate RETURN. Quindi assegna alla variabile RISP$ i caratteri che avete scritto. Troverete più informazioni su LINE INPUT nel Capitolo 10, ma per ora assumete che RISP$ contenga la lettera che avete scritto.
Le linee 30 e 40 contengono istruzioni IF THEN. L'istruzione IF THEN è chiamata istruzione di salto condizionato, poiché l'MSX-BASIC salta ad un'altra linea solo se incontra una determinata condizione. La linea 30 dice all'MSX-BASIC, "Se la persona batte una S, scavalca tutte le linee fino alla linea 70". L'istruzione GOTO dice all'MSX-BASIC dove iniziare ad eseguire le linee, se la condizione tra l'IF e il THEN è vera. La logica seguita dal programma è mostrata nella Figura 7.2. In pratica la linea 30 controlla se battete una S e, se lo fate, vi manda alla linea 70; la linea 40 controlla se battete una N e, se lo fate, vi manda alla linea 90; se non si verificano le condizioni delle linee 30 e 40, andate alla linea 50.
Potete usare un GOTO senza un IF, come potete vedere alle linee 60 e 80: questo si chiama salto incondizionato. La ragione per cui usate il GOTO

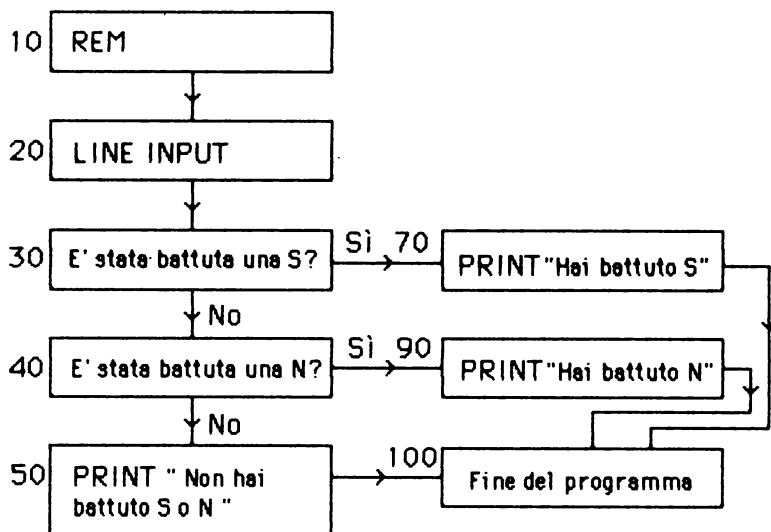



Figura 7.1 Diagramma schematico del programma SI O NO

di linea 60 è per non eseguire la linea 70 dopo la 50. Dopo aver scritto "Non hai battuto S o N", il programma è finito e non volete stampare altro. Quindi, dovete trovare il modo per evitare le linee che stampano "Hai battuto S" e "Hai battuto N"; il salto incondizionato aggira queste linee fino alla linea 100.

La linea 100 contiene l'istruzione END che non fa altro che dire all'MSX-BASIC di fermare il programma. Ricorderete che il programma PRIMO non finiva con END. Niente di strano, poiché l'MSX-BASIC sa che deve fermarsi quando termina la sequenza dei comandi da eseguire. In questo programma, comunque, volevate un punto dove mandare il programma dopo che aveva stampato uno dei tre messaggi. L'istruzione END è il modo formalmente corretto per concludere ogni programma, ma non è indispensabile.

Come vedrete nel Capitolo 9, in un programma i GOTO possono anche mandarvi a linee precedenti. Un programma più lungo può contenere dozzine di istruzioni GOTO che rimandano a tutte le parti del programma.

Capitolo

Variabili, costanti e funzioni

8

Questo è un capitolo molto importante e denso di argomenti. Una volta finito questo capitolo, saprete quasi tutto sulla sintassi dell'MSX-BASIC e conoscerete la maggior parte del suo vocabolario. Le variabili e le costanti costituiscono gli ultimi concetti di fondo che dovete imparare in MSX-BASIC perché sono molto usate nelle istruzioni. Le funzioni sono simili ai comandi, ma diversamente da questi, non appaiono all'inizio di una linea.

8.1 Variabili e costanti

Una variabile è un "posto" dove potete memorizzare un dato. Quel dato può essere sia un numero, sia una serie di lettere. Per esempio, nel programma PRIMO, TIME e I erano variabili che assumevano valore numerico (TIME è una variabile speciale, come vedrete più avanti in questo capitolo). Una variabile che contiene un numero è detta variabile numerica. Nel programma SI O NO, RISP$ è una variabile che contiene dei caratteri. In BASIC, un gruppo di caratteri è chiamato *stringa*; quindi RISP$ è una variabile stringa.
Potete pensare a una variabile come a una casella delle lettere. La scritta sulla casella è il nome che date alla variabile. I dati possono essere messi nella casella e possono essere estratti e letti. Potete cambiare i dati mettendo nuovi dati nella casella delle lettere. Se avete familiarità con l'algebra elementare, sappiate che le variabili in MSX-BASIC sono identiche alle variabili algebriche.

Una costante è invece qualcosa il cui valore non cambia. Per esempio, il numero 15 è una costante numerica: 15 vuol sempre dire 15. Quando definite esplicitamente un testo, avete una costante stringa. Nel programma SI O NO, la stringa alla linea 90, che dice "Hai battuto N", è una costante stringa. Le costanti stringa sono sempre delimitate da virgolette. Le costanti non hanno un nome ma solo un valore.
L'MSX-BASIC vi permette di decidere i nomi per le variabili. Il nome di una variabile può avere una qualsiasi lunghezza, ma l'MSX-BASIC considera solo i primi due caratteri. Il primo carattere deve essere una lettera, mentre l'altro può essere sia un numero che una lettera, ma non un segno di punteggiatura.

TIPI DI VARIABILI E DI COSTANTI

Esistono quattro tipi di variabili e di costanti in MSX-BASIC: interi, reali in precisione semplice, reali in doppia precisione e stringhe. Ricordate che un numero reale è un numero che comprende cifre decimali. L'unica differenza tra i reali in semplice e in doppia precisione è il grado di approssimazione (6 cifre o 14 cifre dopo la virgola) e la quantità di memoria occupata (2 byte e 4 byte).
L'MSX-BASIC pone dei limiti al valore che un numero può assumere. Gli interi possono essere qualsiasi numero tra −32768 e 32767. I numeri reali devono essere tra 10^{-64} e 10^{63}.
Le stringhe occupano in memoria un byte per carattere. Poiché la memoria del computer si basa sui numeri e non sul testo, a ogni carattere corrisponde un numero secondo il codice ASCII. L'Appendice D mostra i codici ASCII dei caratteri che il vostro computer può generare.

Tipo di variabile	Simbolo	Esempi
Intera	%	DELTA%, A%
Reale-precisione semplice	!	GG1!, ALTEZZA!
Reale-doppia precisione	# o niente	PINCO#, LARGH
Stringa	$	RISP$, A1$

Figura 8.1 Simboli dei tipi di variabili

ABS	DSKI$	LOC	RESUME
AND	DSKO$	LOCATE	RETURN
ASC	ELSE	LOF	RIGHT$
ATN	END	LOG	RND
ATTR$	EOF	LPOS	RSET
AUTO	EQV	LPRINT	RUN
BASE	ERASE	LSET	SAVE
BEEP	ERL	MAX	SCREEN
BIN$	ERR	MERGE	SET
BLOAD	ERROR	MID$	SGN
BSAVE	EXP	MKD$	SIN
CALL	FIELD	MKI$	SOUND
CDBL	FILES	MKS$	SPACE$
CHR$	FIX	MOD	SPC
CINT	FN	MOTOR	SPRITE
CIRCLE	FOR	NAME	SQR
CLEAR	FPOS	NEW	STEP
CLOAD	FRE	NEXT	STICK
CLOSE	GET	NOT	STOP
CLS	GO	OCT$	STR$
CMD	GOSUB	OFF	STRIG
COLOR	GOTO	ON	STRING$
CONT	HEX$	OPEN	SWAP
COPY	IF	OR	TAB
COS	IMP	OUT	TAN
CSAVE	INKEY$	PAD	THEN
CSNG	INP	PAINT	TIME
CSRLIN	INPUT	PDL	TO
CVD	INSTR	PEEK	TROFF
CVI	INT	PLAY	TRON
CVS	IPL	POINT	USING
DATA	KEY	POKE	USR
DEF	KILL	POS	VAL
DEFDBL	LEFT$	PRESET	VARPTR
DEFINT	LEN	PRINT	VDP
DEFSNG	LET	PSET	VPEEK
DEFSTR	LFILES	PUT	VPOKE
DELETE	LINE	READ	WAIT
DIM	LIST	REM	WIDTH
DRAW	LLIST	RENUM	XOR
DSKF$	LOAD	RESTORE	

Figura 8.2 Parole riservate MSX-BASIC

NOMI E VALORI DI UNA VARIABILE

L'MSX-BASIC, mentre esegue il vostro programma, distingue che tipo di variabile state usando, guardando il simbolo alla fine del nome della variabile stessa.
La Figura 8.1 mostra il carattere finale per ogni tipo di variabile. Notate che se non mettete un simbolo alla fine del nome di una variabile, l'MSX-BASIC considererà la variabile da voi usata come del tipo in doppia precisione.
C'è un'altra regola che dovete ricordare quando scrivete un programma: le variabili non possono avere un nome uguale alle parole riservate che costituiscono il vocabolario dell'MSX-BASIC. Questa regola evita che l'MSX-BASIC prenda per un comando o una funzione, una variabile che state usando. Inoltre, i nomi delle variabili non possono neppure contenere nel loro interno alcun comando MSX-BASIC. La Figura 8.2 elenca le parole riservate.
Date un'occhiata di nuovo al programma SI O NO:

```
10 REM Esempio dell'uso di GOTO
20 LINE INPUT "Batti S o N: "; RISP$
30 IF RISP$ = "S" THEN GOTO 70
40 IF RISP$ = "N" THEN GOTO 90
50 PRINT "Non hai battuto S o N"
60 GOTO 100
70 PRINT "Hai battuto S"
80 GOTO 100
90 PRINT "Hai battuto N"
100 END
```

Alla linea 20, "Batti S o N" è una costante e RISP$ è una variabile stringa (ora vedete perché il simbolo di dollaro è stato usato nel programma). Quando viene eseguito il comando LINE INPUT, l'MSX-BASIC controlla la tastiera finché premete il tasto RETURN e poi pone i caratteri che avete battuto nella variabile stringa da voi nominata. Dovete usare una variabile stringa con l'istruzione LINE INPUT. Alla linea 30, RISP$ è, naturalmente, ancora una variabile e S è una costante. Il comando IF THEN confronta il contenuto della variabile RISP$ con la costante S.
La maggior parte delle istruzioni che avete visto, oltre a costanti, possono anche utilizzare le variabili. Per esempio, il seguente programma funziona esattamente come il programma "SI O NO":

```
10 REM Esempio dell'uso di variabili
20 REM al posto delle costanti
30 REM nel programma SI O NO
40 SRISP$ = "S"
50 NRISP$ = "N"
```

```
60 NES$ = "Non hai battuto S o N"
70 SMES$ = "Hai battuto S"
80 NMES$ = "Hai battuto N"
90 LINE INPUT "Batti S o N: "; RISP$
100 IF RISP$ = SRISP$ THEN GOTO 140
110 IF RISP$ = NRISP$ THEN GOTO 160
120 PRINT NES$
130 GOTO 170
140 PRINT SMES$
150 GOTO 170
160 PRINT NMES$
170 END
```

Notate che le variabili SRISP$, NRISP$, NES$, SMES$ e NMES$ vengono definite nelle linee dalla 40 all'80 e usate poi nel resto del programma. Questo non è il metodo standard di programmazione, ma è un buon esempio di come potete usare le variabili in molte parti del vostro programma.
L'errore più comune che potete commettere scrivendo programmi in MSX-BASIC è quello di dimenticare che esso considera solo le prime due lettere del nome della variabile. Dovete tenere in mente questa regola quando date il nome ad una nuova variabile. Per esempio, osservate il seguente semplice programma:

```
10 REM Esempio di errore comune
20 REM nei nomi delle variabili
30 VAR1 = 5
40 VAR2 = 6
50 PRINT VAR1
60 PRINT VAR2
```

A prima vista, vi potreste aspettare che il programma stampi 5 e poi 6. Invece, esso stampa 6 e poi ancora 6, poiché l'MSX-BASIC riconosce solo i primi due caratteri dei nomi delle variabili.
Quando l'MSX-BASIC esegue il programma è come se aveste scritto

```
10 REM Interpretazione dell'errore
20 REM nei nomi delle variabili
30 VA = 5
40 VA = 6
50 PRINT VA
60 PRINT VA
```

Alla linea 50, VA è uguale a 6.
Se vi sembra noioso usare i caratteri identificatori di tipo alla fine del nome delle variabili, potete usare il comando DEF per dire all'MSX-BASIC che tutte le variabili che iniziano con una certa lettera, rappresen-

tano un determinato tipo di variabile. I comandi sono DEF DBL, DEF INT, DEF SNG, DEF STR che stanno per reali a doppia precisione, interi, reali a singola precisione e stringhe. Potete dare un comando con una lettera o con un insieme di lettere. Per esempio,

```
10 DEF SNG C-E
20 DEF INT I
```

dice all'MSX-BASIC che tutte le variabili la cui prima lettera è una C, una D o una E sono numeri reali a singola precisione e tutte le variabili il cui nome inizia per I sono interi.
Ci sono variabili speciali che l'MSX-BASIC definisce per voi. Queste verranno menzionate nei capitoli seguenti, ma ne avete già incontrata una: TIME. TIME è una variabile numerica che viene incrementata di 1 ogni 1/50 di secondo. Potete inizializzare il valore di TIME in qualsiasi punto del vostro programma. Questo spiega l'uso di TIME nel programma PRIMO: l'avete inizializzata a 0 e poi avete atteso che diventasse 100. Siccome viene aggiornata ogni 1/50 di secondo, questo significa attendere 2 secondi. Le linee 50 e 60 di quel programma sono:

```
50 TIME = 0
60 IF TIME < 100 THEN GOTO 60
```

Quando l'MSX-BASIC raggiunge la linea 60, controlla se il valore di TIME è inferiore a 100. Dato che il vostro computer MSX funziona molto velocemente, la prima volta che l'MSX-BASIC arriva alla linea 60, TIME è probabilmente uguale a 0. Quindi, l'MSX-BASIC va alla linea 60 e riesegue la linea. Finché il valore di TIME non è incrementato a 100, l'MSX-BASIC continua ad eseguire la linea 60. Quando TIME è uguale a 100, l'MSX-BASIC trova che la condizione TIME<100 è falsa e passa alla linea 70.
Se non vi è chiaro il funzionamento di TIME, eseguite il programma che segue:

```
0  REM
10 REM Esperimento con TIME
20 PRINT "Parte il conteggio"
30 TIME = 0
40 PRINT TIME
50 REM Sono trascorsi 5 secondi?
60 IF TIME < 250 GOTO 40
70 PRINT "Sono trascorsi 5 secondi"
```

GLI ARRAY

In molti casi, è utile avere una serie di variabili cui poter accedere con un solo nome. Un array è proprio una serie di variabili. Potete avere array di stringhe, di reali in singola o in doppia precisione o di interi. Ogni elemento nell'array ha un proprio valore e può essere trattato come una variabile separata.
Un array si definisce con il comando DIM. La forma più semplice di questo comando è

DIM *nomevar* (*n*)

dove *nomevar* è il nome che volete dare al vostro array e *n* è il numero massimo di elementi nell'array. Ogni array inizia la propria numerazione a partire da 0, quindi con il comando DIM create un array di $n+1$ elementi. Per esempio, per definire un array di 15 elementi, chiamato LL, potete dare il comando

```
10 DIM LL(14)
```

Ora, nel vostro programma in MSX-BASIC, potete dare dei comandi che utilizzano queste variabili, come

```
50 LL(5) = 23.5
60 N = 7
70 LL(N) = 9.1
```

La linea 50 assegna un valore alla variabile LL(5), mentre la linea 70 ne assegna uno alla variabile LL(7).
Questi sono array con una sola serie di elementi, conosciuti anche come array monodimensionali. Si possono avere anche array a più dimensioni, semplicemente specificandole nel comando DIM. Per esempio, per creare un array chiamato QN che abbia 12 righe e 17 colonne, date il comando

```
120 DIM QN(12,17)
```

Per accedere ad una singola variabile in questo array, date entrambe le coordinate:

```
180 QN(7,5) = 3
```

NUMERAZIONI

L'MSX-BASIC può gestire le costanti in una quantità di numerazioni diverse. La maggior parte delle persone ha familiarità con la sola numerazione decimale ma molti conoscono anche la numerazione scientifica, anche nota come forma esponenziale. Nella numerazione scientifica, un numero ha nella parte sinistra un intero o un reale (detto *mantissa*), quindi una E e poi un intero (l'esponente). L'MSX-BASIC interpreta questo numero come il prodotto di 10 elevato all'esponente, per la mantissa.
Per esempio, il numero 4.732E5 è uguale a 473200. Con la E un numero in numerazione scientifica viene considerato in singola precisione. Per specificare un numero in doppia precisione, sostituite la E con una D (ad esempio 4.732D5).
Siccome i computer lavorano in numerazione binaria, è spesso utile esprimere i numeri destinati al computer in numerazione binaria, ottale o esadecimale.
Se volete esprimere delle costanti in numerazione binaria, anteponete un &B all'inizio del numero. Per specificare l'ottale usate &O, e per l'esadecimale usate &H. Le linee seguenti assegnano il numero 431 (decimale) alla variabile VG:

```
VG = &B110101111
VG = &O657
VG = &H1AF
```

8.2 Funzioni

Una funzione è un'operazione che elabora un numero o una stringa. Le funzioni si possono considerare dei sottoprogrammi e assomigliano alle funzioni matematiche. Esse hanno argomenti proprio come i comandi, ma questi sono posti tra parentesi a destra del nome della funzione. Alcune funzioni hanno un solo argomento, altre ne hanno di più.
Per esempio, ricorderete probabilmente dal liceo, la radice quadrata. La radice di un numero è a sua volta un numero; quest'ultimo, moltiplicato per se stesso, riproduce il numero iniziale. Nell'MSX-BASIC, per calcolare la radice quadrata di un numero si usa la funzione SQR. La sintassi della funzione SQR è

SQR (*n*)

dove *n* è il numero di cui volete trovare la radice quadrata. Per esempio:

```
30 B = SQR(A)
```

Funzione	Significato
ABS (*numero*)	Valore assoluto di *numero*
ATN (*numero*)	Arcotangente in radianti
COS (*numero*)	Coseno in radianti
EXP (*numero*)	Esponenziale (e^{numero})
FIX (*numero*)	Toglie la parte frazionaria di *numero*
INT (*numero*)	Arrotonda *numero* per difetto
LOG (*numero*)	Logaritmo naturale di *numero*
SGN (*numero*)	Restituisce +1 se *numero* > 0 0 se *numero* = 0 -1 se *numero* < 0
SIN (*numero*)	Seno in radianti
SQR (*numero*)	Radice quadrata
TAN (*numero*)	Tangente in radianti

Figura 8.3 Funzioni matematiche dell'MSX-BASIC

Qui, B viene posto uguale alla radice quadrata di A.
Potete usare una funzione ovunque potete usare una costante, poiché una funzione ha sempre un valore, proprio come una costante. Potete usare quindi una funzione all'interno di una funzione. Per esempio, se volete inizializzare J1 al logaritmo del seno di J2, potete dare il comando

```
50 J1 = LOG(SIN(J2))
```

Quando l'MSX-BASIC vede questo, prima calcola SIN (J2), e poi il logaritmo del risultato.
L'MSX-BASIC ha molte funzioni matematiche per ogni esigenza. Sono elencate nella Figura 8.3. Ci sono inoltre molte funzioni stringa, elencate nella Figura 8.4. La Figura 8.5 poi, elenca le funzioni che vi permettono di convertire i numeri in stringhe e viceversa.
Sebbene molte di queste funzioni si spieghino da sole, un esempio può servire a chiarirne l'uso.

```
10 REM Esempi di funzioni MSX-BASIC
20 REM Prima le funzioni numeriche
30 A = 5
40 B = 10
50 C = -3.7
```

```
60 D = .8245
70 PRINT "Il valore assoluto di "; C; "e'"; ABS(C)
80 PRINT "L' arcotangente di "; A; "e'"; ATN(A)
90 PRINT "Il coseno di"; D; "e'"; COS(D)
100 PRINT "L' esponenziale di"; A; "e'"; EXP(A)
110 PRINT "Il FIX di"; C; "e'"; FIX(C)
120 PRINT "Il FIX di"; B; "e'"; FIX(B)
130 PRINT "L' INT di"; B; "e'"; INT(B)
140 PRINT "L' INT di"; C; "e'"; INT(C)
150 PRINT "Il logaritmo di"; A; "e'"; LOG(A)
160 PRINT "Il segno di"; A; "e'"; SGN(A)
170 PRINT "Il segno di"; C; "e'"; SGN(C)
180 PRINT "Il seno di"; D; "e'"; SIN(D)
190 PRINT "La radice quadrata di"; A; "e'"; SQR(A)
200 PRINT "La tangente di"; D; "e'"; TAN(D)
210 REM Ora le funzioni stringa
220 X$ = "donna di quadri"
230 LUNG = 6
240 PART = 4
250 STRLUN$ = "tanto va la gatta al lardo"
260 STRCO$ = "lei"
270 CARATTERE$ = "g"
280 PRINT "INSTR da' ***"; INSTR(PART, STRLUN$,
STRCO$); "***"
290 PRINT "LEFT$ da' ***"; LEFT$(X$, LUNG);
"***"
300 PRINT "LEN da' ***"; LEN(X$); "***"
310 PRINT "MID$ da' ***"; MID$(X$, PART, LUNG);
"***"
320 PRINT "RIGHT$ da' ***"; RIGHT$(X$, LUNG);
"***"
330 PRINT "SPACE$ da' ***"; SPACE$(LUNG); "***"
340 PRINT "STRING$ da' ***"; STRING$(LUNG,
CARATTERE$); "***"
350 REM Infine, le funzioni di conversione
360 X$ = "d"
370 PRINT "Il valore ASCII di "; X$; " e'"; ASC(X$)
380 NUM = 100
390 PRINT "Il valore binario di"; NUM; "e'";
BIN$(NUM)
400 PRINT "CHR$ da' ***"; CHR$(NUM);"***"
410 PRINT "Il valore esadecimale di"; NUM; "e'";
HEX$(NUM)
420 PRINT "Il valore ottale di"; NUM; "e'"; OCT$(NUM)
430 PRINT "Il valore decimale di"; NUM; "e'";
STR$(NUM)
440 X$ = "-34.99"
450 PRINT "VAL da' ***"; VAL(X$)
```

Questo programma stampa ciò che segue (cercate di capire il perché):

```
Il valore assoluto di -3.7 e' 3.7
L' arcotangente di  5 e' 1.373400766945
Il coseno di .8245 e' .67892415468003
L' esponenziale di 5 e' 148.41315910255
Il FIX di-3.7 e'-3
Il FIX di 10 e' 10
L' INT di 10 e' 10
L' INT di-3.7 e'-4
Il logaritmo di 5 e' 1.6094379124341
Il segno di 5 e' 1
Il segno di-3.7 e'-1
Il seno di .8245 e' .73420841195945
La radice quadrata di 5 e' 2.2360679774998
La tangente di .8245 e' 1.0814292095786
INSTR da' *** 0 ***
LEFT$ da' ***donna ***
LEN da' *** 15 ***
MID$ da' ***na di ***
RIGHT$ da' ***quadri***
SPACE$ da' ***       ***
STRING$ da' ***gggggg***
Il valore ASCII di d e' 100
Il valore binario di 100 e'1100100
CHR$ da' ***d***
Il valore esadecimale di 100 e'64
Il valore ottale di 100 e'144
Il valore decimale di 100 e' 100
VAL da' ***-34.99
```

Ci sono altre due funzioni che non sono facili da descrivere quanto quelle già mostrate. Queste due funzioni sono MID$= e RND.
La funzione MID$= è diversa dalla funzione MID$, sebbene abbia molto in comune con questa. MID$= è la sola funzione che può apparire nella parte sinistra di un comando LET. Ricordate che il comando LET (con LET esplicitato o no) assegna alle variabili a sinistra del segno di uguale il valore alla destra di quest'ultimo. Se usate la funzione MID$ a sinistra del segno di uguale, è perché avete intenzione di dire all'MSX-BASIC di cambiare il valore di una parte di una stringa.
Il programma che segue mostra come usare la funzione MID$=

```
10 REM Esempio di MID$=
20 ASTRINGA$ = "aaaaaaaaaa"
30 BSTRINGA$ = "bbbbbbbbbb"
40 MID$(ASTRINGA$,3,4) = BSTRINGA$
50 PRINT ASTRINGA$
```

Questo programma stampa:

```
aabbbbaaaa
```

La funzione RND genera un numero casuale tra 0 e 1. Ciò è utile se volete simulare un evento casuale, come il tiro di un dado. La funzione RND accetta diversi argomenti che danno risultati diversi:

- Qualsiasi numero positivo genera un numero casuale in doppia precisione, tra 0 e 1.
- Qualsiasi numero negativo serve a far iniziare una nuova serie di nu-

Funzione	**Significato**
INSTR(*inizio*, *stringa1*, *stringa2*)	Partendo dalla posizione *inizio* (opzionale), trova la prima occorrenza di *stringa2* in *stringa1*. Se *stringa2* non è contenuta in *stringa1*, il risultato è 0
LEFT$(*stringa*, *lungh*)	Restituisce la parte sinistra di lunghezza *lungh* di *stringa*
LEN(*stringa*)	Restituisce la lunghezza di *stringa*
MID$(*stringa*, *inizio*, *lungh*)	Restituisce una sottostringa di *stringa* di lunghezza *lungh*, a partire da *inizio* (opzionale)
RIGHT$(*stringa*, *lungh*)	Restituisce la parte destra di lunghezza *lungh* di *stringa*
SPACE$(*lungh*)	Crea una stringa di spazi di lunghezza *lungh*
STRING$(*lungh*, *car*)	Crea una stringa di caratteri uguali a *car* e di lunghezza *lungh*

Figura 8.4 Funzioni stringa dell'MSX-BASIC

Funzione	Significato
ASC(*stringa*)	Codice ASCII del primo carattere di *stringa*
BIN$(*numero*)	Stringa che rappresenta *numero* in codice binario
CDBL(*numero*)	Reale in doppia precisione
CHR$(*numero*)	Carattere il cui codice ASCII è *numero*
CINT(*numero*)	Intero
CSNG(*numero*)	Numero in precisione semplice
HEX$(*numero*)	Stringa che rappresenta *numero* in esadecimale
OCT$(*numero*)	Stringa che rappresenta *numero* in ottale
STR$(*numero*)	Stringa che rappresenta *numero* in decimale
VAL(*stringa*)	Reale in doppia precisione che rappresenta *stringa*, quando questa è composta da cifre

Figura 8.5 Funzioni di conversione dell'MSX-BASIC

meri al generatore di numeri causali; questa è detta inizializzazione. Se non inizializzate il generatore di numeri ogni volta che eseguite un programma, esso produce sempre gli stessi numeri casuali (che, naturalmente, di casuale avrebbero ben poco).

- Se come argomento usate lo 0, il generatore vi darà l'ultimo numero casuale prodotto.

Il seguente programma mostra come usare i numeri casuali:

```
10 REM Esempio di numeri casuali
20 FOR I = 1 TO 10
30 PRINT RND(5)
40 NEXT I
50 REM Reinizializzazione del generatore
60 X = RND(-TIME)
70 FOR I = 1 TO 10
80 PRINT RND(5)
90 NEXT I
```

Siccome la variabile TIME cambia ogni 1/50 di secondo, inizializzare il generatore di numeri casuali con RND(−TIME), è un modo sicuro per generare numeri casuali.

8.3 Aritmetica con l'MSX-BASIC

L'MSX-BASIC è spesso usato per effettuare dei calcoli e può svolgere infatti un gran numero di operazioni matematiche. Queste operazioni ricadono in due categorie, numerica e logica. Le operazioni numeriche sono quelle che avete imparato alla scuola elementare (addizione, sottrazione, eccetera). Le operazioni logiche sono quelle che servono a confrontare due elementi e determinare se il confronto è vero o falso.
La Figura 8.6 elenca le operazioni numeriche. Come potete vedere, la maggior parte di esse sono abbastanza semplici e probabilmente vi sono familiari. Gli operatori con i quali le persone hanno meno familiarità sono quello di divisione intera, resto di una divisione ed elevazione a potenza. Quando si usa l'operatore di divisione intera, si ottiene sempre un risultato intero. Il risultato di una divisione intera è lo stesso di una divisione reale, ma viene tolta la parte frazionaria. Il resto intero (MOD) è il resto della divisione di due interi. Quando l'operatore di elevazione a potenza viene posto tra due numeri, il risultato è il primo numero elevato al secondo.

Operatore	Operazione
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
\	Divisione intera
MOD	Resto di una divisione intera
^	Elevazione a potenza

Figura 8.6 Operatori matematici

Le operazioni numeriche sono mostrate nel seguente programma:

```
10 REM Esempio di operazioni
20 X = 3.4
30 Y = 6.6
40 PRINT X+Y
50 PRINT X-Y
60 PRINT X*Y
70 PRINT X/Y
```

```
80 Z1 = 64
90 Z2 = 15
100 PRINT Z1\Z2
110 PRINT Z1 MOD Z2
120 PRINT X^Y
```

Il programma stampa

```
 10
-3.2
 22.44
 .51515151515152
 4
 4
 3219.295573802
```

Potete usare parentesi per unire assieme parti di espressioni matematiche. Le parentesi nell'MSX-BASIC sono identiche a quelle usate in matematica: le operazioni all'interno delle parentesi sono eseguite separatamente. Nell'esempio che segue MAXVAL viene posto uguale a 45:

```
240 MAXVAL = 5*((1+2)^2)
```

Come potete vedere, l'MSX-BASIC vi permette di usare parentesi all'interno di altre parentesi.
Se non usate le parentesi, l'MSX-BASIC considera gli operatori secondo determinate regole di priorità. Ciò significa che certe operazioni vengono svolte prima di altre; per esempio, la moltiplicazione viene eseguita prima della addizione. Come potete immaginare, cercare di ricordare la priorità in una linea come quella che segue, può essere difficile:

```
40 QUAL = 93.4+73/72.8*8-3
```

Operatore	**Relazione**
=	Uguale
<>	Diverso
<	Minore
<=	Minore o uguale
>	Maggiore
>=	Maggiore o uguale

Figura 8.7 Operatori di relazione

I buoni programmatori usano sempre le parentesi, e questa è un'abitudine che dovreste prendere anche voi.
Gli operatori di relazione vengono usati per confrontare due numeri o due stringhe. Sono usati principalmente nelle istruzioni IF THEN. Finora avete visto due operatori di relazione: "<" usato alla linea 60 di PRIMO, e "=" usato alla linea 30 di SI O NO. Quando l'MSX-BASIC valuta una relazione, il risultato è 0 se la relazione è vera e qualsiasi numero diverso da zero (di solito −1) se la relazione è falsa. La Figura 8.7 elenca gli operatori di relazione usati dall'MSX-BASIC. L'MSX-BASIC ha anche operatori logici, che sono simili agli operatori di relazione. Potete usare gli operatori logici tra due valori logici (cioè, tra due valori che sono veri o falsi). Gli operatori logici seguono la definizione standard per i confronti, come mostrato in Figura 8.8.

X	Y	X AND Y	X OR Y	X EQV Y	X IMP Y	X XOR Y
1	1	1	1	1	1	0
1	0	0	1	0	0	1
0	1	0	1	0	1	1
0	0	0	0	1	1	0

Figura 8.8 Operatori logici

L'operatore NOT è unitario: esso opera solo sull'argomento che lo segue. Come potete aspettarvi, NOT dà l'opposto dell'argomento (falso diventa vero e vero diventa falso). Il programma che segue mostra gli operatori logici e di relazione; esso utilizza una forma di IF THEN che non avete ancora visto: IF THEN ELSE. Se la condizione tra IF e THEN è vera, l'istruzione dopo THEN viene eseguita; se la condizione è falsa, viene invece eseguita l'istruzione dopo l'ELSE.

```
10 REM Operatori logici
20 REM e di relazione
30 P = 10
40 Q = 20
50 PRINT "P = Q e' ";
60 IF P = Q THEN PRINT "vero" ELSE PRINT "falso"
70 PRINT "P <> Q e' ";
80 IF P <> Q THEN PRINT "vero" ELSE PRINT "falso"
90 PRINT "P < Q e' ";
100 IF P < Q THEN PRINT "vero" ELSE PRINT "falso"
```

```
110 PRINT "P <= Q e' ";
120 IF P <= Q THEN PRINT "vero" ELSE PRINT "falso"
130 PRINT "P > Q e' ";
140 IF P > Q THEN PRINT "vero" ELSE PRINT "falso"
150 PRINT "P >= Q e' ";
160 IF P >= Q THEN PRINT "vero" ELSE PRINT "falso"
170 REM Inizializza VE a vero, e FA a falso
180 VE = 0
190 FA = 1
200 PRINT "Vero AND falso da' "
210 IF VE AND FA THEN PRINT "vero" ELSE PRINT "falso"
220 PRINT "Vero XOR falso da' "
230 IF VE XOR FA THEN PRINT "vero" ELSE PRINT "falso"
```

Potete sostituire i valori in questo programma per verificare le altre operazioni logiche.

8.4 Manipolazione delle stringhe

Proprio come potete sommare tra loro e confrontare l'un l'altro i numeri, potete fare queste stesse operazioni con le stringhe. Quando sommate assieme due stringhe (questa operazione è detta *concatenazione*), l'MSX-BASIC crea una nuova stringa con quelle due, poste una di seguito all'altra. Per esempio,

```
10 REM Concatenazione di stringhe
20 K$ = "Liberta' "
30 L$ = "di scelta"
40 M$ = K$ + L$
50 PRINT M$
```

produce la scritta

```
Liberta' di scelta
```

Con le stringhe potete anche usare gli operatori di relazione che avete usato tra i numeri. L'MSX-BASIC confronta due stringhe un carattere per volta, a partire da quello più a sinistra. Tutti i confronti si basano sul valore ASCII dei caratteri. Così il confronto

"Abc" < "a"

risulterebbe vero, poiché "A" ha valore ASCII minore di "a" e l'MSX-BASIC non considera la lunghezza della stringa. Se due stringhe hanno gli stessi caratteri, ma una è più corta, quest'ultima è la minore.
Ora che avete visto tutti i modi in cui potete usare le variabili e fare della matematica con l'MSX-BASIC, siete pronti per iniziare a scrivere programmi. I capitoli che seguono trattano dei comandi e delle funzioni che vi aiuteranno a scrivere programmi che vadano oltre il semplice calcolo numerico.

Capitolo

Il controllo del flusso del programma

Nel Capitolo 7 abbiamo visto alcune istruzioni che influenzano il flusso di un programma; questo capitolo spiega molti altri modi per controllare un programma. Le istruzioni che avete usato finora per evitare la semplice lettura in sequenza delle linee, da parte dell'MSX-BASIC, sono IF THEN, IF THEN ELSE e GOTO. Avete anche visto il comando FOR nel programma PRIMO. Questo capitolo tratta tutti questi comandi ed alcuni comandi più avanzati.

9.1 Le istruzioni IF THEN e IF THEN ELSE

Più programmi scriverete, più spesso vi ritroverete ad usare IF THEN e IF THEN ELSE. C'è una semplicissima ragione alla base di ciò: tutti i programmi veramente utili devono prendere decisioni in base a valori che possono variare.
IF THEN ELSE costituisce un caso particolare dell'IF THEN. In un'istruzione IF THEN senza ELSE, l'MSX-BASIC esegue la linea successiva se la condizione considerata è falsa; nell'istruzione IF THEN ELSE, l'MSX-BASIC esegue l'istruzione dopo l'ELSE se la condizione è falsa. Poiché queste due istruzioni sono praticamente uguali, le vedremo insieme.
Ricordate che, sebbene sia permessa una sola condizione tra IF e THEN, questa può essere molto complessa. In questo caso potete usare delle parentesi per evitare che l'MSX-BASIC si confonda. Per esempio la linea seguente controlla cinque variabili:

```
130 IF (((GIORNO$ = "12") AND (MESE$ = "Gennaio")
AND (ANNO% =  1986)) XOR (NOT(V1 <= V2))) THEN GOTO
190
```

Ci sono due modi fondamentali di usare l'istruzione IF THEN:

IF *condizione* THEN *azione*
IF *condizione* THEN GOTO *numerolinea*

Usate il primo modo se volete fare una cosa sola quando *condizione* è verificata, usate il secondo per tutti gli altri casi.
Un uso comune di IF THEN con un parametro del tipo *azione*, consiste nel cambiare una variabile in base al suo stesso valore. Per esempio, se richiedete un input in lettere maiuscole, ma l'utente immette una lettera minuscola, potreste modificare la variabile per avere sempre una risposta in maiuscolo. Quella che segue è una variazione del programma SI O NO che accetta "N", "n", "S", "s":

```
10 REM Nuovo esempio dell' uso dei salti
20 LINE INPUT "Batti S o N: "; RISP$
22 IF RISP$ = "s" THEN RISP$ = "S"
24 IF RISP$ = "n" THEN RISP$ = "N"
30 IF RISP$ = "S" THEN GOTO 70
40 IF RISP$ = "N" THEN GOTO 90
50 PRINT "Non hai battuto S o N"
60 GOTO 100
70 PRINT "Hai battuto S"
80 GOTO 100
90 PRINT "Hai battuto N"
100 END
```

Le linee 22 e 24 sono state aggiunte per poter usare anche le lettere minuscole. Se seguite il flusso di questo programma, noterete che non è molto efficiente. Per esempio, se l'utente batte una "s", l'MSX-BASIC se ne accorge alla linea 22; è inutile eseguire le linee 24 e 30 prima di andare alla linea 70. In questo caso, è probabilmente una buona idea aggiungere un'altra istruzione alle linee 22 e 24 dopo l'istruzione IF THEN. Ricordate che il modo per far questo consiste nel porre un due punti (:) tra le due istruzioni.
Se IF THEN ha una seconda istruzione dopo di sé, quel comando sarà eseguito dopo l'argomento *azione*, solo se la condizione è vera. Il nuovo programma ora è:

```
10 REM Nuovo esempio dell' uso dei salti
20 LINE INPUT "Batti S o N: "; RISP$
22 IF RISP$ = "s" THEN RISP$ = "S" : GOTO 70
24 IF RISP$ = "n" THEN RISP$ = "N" : GOTO 90
```

```
30 IF RISP$ = "S" THEN GOTO 70
40 IF RISP$ = "N" THEN GOTO 90
50 PRINT "Non hai battuto S o N"
60 GOTO 100
70 PRINT "Hai battuto S"
80 GOTO 100
90 PRINT "Hai battuto N"
100 END
```

Ora la linea 22 dice all'MSX-BASIC che se RISP$ è uguale a "s", deve cambiare il suo valore in "S" e poi che deve saltare alla linea 70. In questo programma, in realtà non avete bisogno di correggere il valore di RISP$ prima di andare alla linea che stampa il messaggio, ma ciò sarebbe stato assolutamente necessario se altre linee fossero dipese dal valore di RISP$. Queste aggiunte rendono il programma più elegante e più veloce.
C'è ancora un altro modo per abbreviare questo programma. La linea 40 può essere modificata in un IF THEN ELSE con l'argomento *azione* e un'altra istruzione; questo può eliminare le linee 50 e 60. Il risultato è

```
10 REM Nuovo esempio dell' uso dei salti
20 LINE INPUT "Batti S o N: "; RISP$
22 IF RISP$ = "s" THEN RISP$ = "S" : GOTO 70
24 IF RISP$ = "n" THEN RISP$ = "N" : GOTO 90
30 IF RISP$ = "S" THEN GOTO 70
40 IF RISP$ = "N" THEN GOTO 90 ELSE PRINT
"Non hai battuto S o N" : GOTO 100
70 PRINT "Hai battuto S"
80 GOTO 100
90 PRINT "Hai battuto N"
100 END
```

Questo programma funziona come il precedente.

QUALCOSA DI PIÙ SULLA MODIFICA DEI PROGRAMMI

La seguente digressione vi darà una visione più approfondita del metodo per modificare i vostri programmi. È opportuna a questo punto perché vi siete appena trovati a dover modificare un programma, aggiungendo e cancellando linee.
Come avete appena visto, aggiungere una linea nuova è semplice: dovete solamente darle un numero compreso tra quelli delle due linee fra le quali volete che appaia. Quando date il comando LIST, la nuova linea appare tra le altre. Per cancellare delle linee usate il comando DELETE. L'argomento del comando DELETE è simile a quello del comando LIST,

cioè un intervallo di numeri di linea. Dopo aver modificato la linea 40, volete cancellare le linee dalla 50 alla 60:

```
DELETE 50-60
```

La versione finale di questo programma permette di introdurre un interessante comando MSX-BASIC, che è utile quando modificate i programmi. Notate che la vostra bella e ordinata numerazione delle linee è stata rovinata dall'aggiunta di alcune righe e dall'eliminazione di altre. Se volete ancora le vostre linee numerate per 10 senza salti, usate il comando RENUM. Controllate poi con LIST: il vostro programma è nuovamente ben numerato.

```
10 REM Nuovo esempio dell' uso dei salti
20 LINE INPUT "Batti S o N: "; RISP$
30 IF RISP$ = "s" THEN RISP$ = "S" : GOTO 70
40 IF RISP$ = "n" THEN RISP$ = "N" : GOTO 90
50 IF RISP$ = "S" THEN GOTO 70
60 IF RISP$ = "N" THEN GOTO 90 ELSE PRINT
"Non hai battuto S o N" : GOTO 100
70 PRINT "Hai battuto S"
80 GOTO 100
90 PRINT "Hai battuto N"
100 END
```

Il comando RENUM è molto potente. Aggiungete la seguente linea al programma:

```
55 REM La prossima linea e' lunga
```

Ora date i comandi RENUM e LIST; il risultato è

```
10 REM Nuovo esempio dell' uso dei salti
20 LINE INPUT "Batti S o N: "; RISP$
30 IF RISP$ = "s" THEN RISP$ = "S" : GOTO 80
40 IF RISP$ = "n" THEN RISP$ = "N" : GOTO 100
50 IF RISP$ = "S" THEN GOTO 80
60 REM La prossima linea e' lunga
70 IF RISP$ = "N" THEN GOTO 100 ELSE PRINT
"Non hai battuto S o N" : GOTO 110
80 PRINT "Hai battuto S"
90 GOTO 110
100 PRINT "Hai battuto N"
110 END
```

Notate che la linea che prima era la linea 60 ora è la 70, la vecchia linea 70 è ora la 80 e così via. Sarete terrorizzati dal pensiero che ora tutti i

GOTO siano sbagliati, ma se confrontate questa versione con la precedente, vedrete che il comando RENUM ha già provveduto a modificare gli indirizzi dei GOTO. Quindi potete usare il comando RENUM ogni volta che volete, sicuri che tutti i vostri numeri di linea, sia quelli all'interno delle istruzioni che quelli all'inizio delle linee, coincideranno correttamente.

9.2 Le istruzioni FOR e NEXT

Ci sono molte occasioni in cui bisogna eseguire ripetutamente una funzione per un certo numero di volte. Per esempio, vi ricorderete di quando in terza elementare avete dovuto scrivere "Non tirerò più palline di carta" per cento volte sulla lavagna. Le istruzioni FOR e NEXT rendono questo compito molto più facile. Fate girare questo programma:

```
10 REM Punizione
20 FOR CN = 1 TO 100
30 PRINT "Non tirero' piu' palline di carta"
40 NEXT CN
50 PRINT "Posso andare a casa adesso?"
```

Le linee scorreranno via rapidamente dallo schermo, ma state tranquilli che l'MSX-BASIC ha scritto la frase 100 volte.
FOR si trova sempre associato a un NEXT: ecco perché le istruzioni e le linee che si trovano tra essi sono chiamate ciclo o loop FOR NEXT. Le linee dalla 20 alla 40 nel precedente programma costituiscono un ciclo FOR NEXT. La sintassi di un ciclo FOR NEXT è

FOR *variabile* = *inizio* TO *fine*
.
.
.
NEXT *variabile*

Dove *variabile* è una variabile numerica e *inizio* e *fine* due interi a piacere. Quando l'MSX-BASIC incontra un FOR pone *variabile* uguale a *inizio* e poi esegue le altre istruzioni come al solito. Quando arriva al NEXT, salta indietro al FOR, aggiunge 1 a *variabile* e controlla se *variabile* è maggiore di *fine*. Se non è maggiore, percorre un'altra volta il ciclo; se è maggiore, salta alla istruzione successiva al NEXT.
Quello che segue è un semplice esempio di loop FOR NEXT. Notate che la variabile H1 viene utilizzata all'interno del loop; ciò è perfettamente accettabile.

```
10 REM Semplice ciclo FOR-NEXT
20 PRINT "Inizio"
30 FOR H1 = 1 TO 5
40 PRINT H1
50 NEXT H1
60 PRINT "Fine"
```

Quando gira, il programma stampa:

```
Inizio
 1
 2
 3
 4
 5
Fine
```

Sebbene sia permesso modificare il valore della variabile del ciclo all'interno del ciclo stesso, non è una buona idea, perché potreste compromettere la corretta esecuzione del loop.
L'MSX-BASIC vi permette di mettere qualsiasi istruzione all'interno di un loop FOR NEXT, anche un altro loop FOR NEXT: in questo caso si parla di annidamento dei loop FOR NEXT. Se annidate dei loop, dovete essere ben certi che il loop più interno sia completamente contenuto in quello più esterno, poiché l'MSX-BASIC dà un messaggio di errore se trova in un comando NEXT il nome della variabile sbagliato.
Il seguente programma disegna un rettangolo con i numeri dal 100 al 148. Potreste credere che sia molto difficile da fare senza un programma elaborato; come potete invece vedere, è facile da realizzare con i loop FOR NEXT annidati.

```
10 REM Crea un rettangolo
20 FOR X = 14 TO 20
30 FOR Y = 0 TO 6
40 PRINT (7*X)+Y+2;
50 NEXT Y
60 PRINT:PRINT
70 NEXT X
```

Nel prossimo capitolo vedrete che il punto e virgola (;) dopo il comando PRINT, fa sì che i numeri vengano stampati sulla stessa linea.
L'istruzione FOR può anche avere l'opzione STEP, che dice all'MSX-BASIC quanto deve aggiungere alla variabile ogni volta che percorre il ciclo. Se usate l'argomento STEP, la sintassi del ciclo FOR NEXT è

FOR *variabile* = *inizio* TO *fine* [STEP *passo*]
.
.
.
NEXT [*variabile*]

L'MSX-BASIC aggiunge *passo* anziché 1 a *variabile* ogni volta che ripercorre il ciclo. Il precedente programma può essere ora riscritto:

```
10 REM Crea un rettangolo
20 FOR X = 100 TO 142 STEP 7
30 FOR Y = 0 TO 6
40 PRINT X+Y;
50 NEXT Y
60 PRINT:PRINT
70 NEXT X
```

Se volete, potete usare un passo negativo, il che vi serve per eseguire dei conti alla rovescia.

```
10 REM Conto alla rovescia
20 FOR Q = 37 TO 28 STEP -1
30 PRINT Q
40 NEXT Q
```

Questo è l'output:

```
37
36
35
34
33
32
31
30
29
28
```

9.3 Le istruzioni GOSUB e RETURN

Una potente caratteristica di molti linguaggi moderni di programmazione è costituita dalle subroutine. Una subroutine è come un mini-programma che si trova all'interno di un programma più grande. È utile quando c'è una serie di passi che volete ripetere più volte in differenti parti di un programma. Anziché ripetere tutti i passi ogni volta, potete creare una subroutine, saltare alla subroutine quando ne avete bisogno, e tornare indietro.

Per entrare in una subroutine si usa l'istruzione GOSUB (*GO to SUBroutine*). Tutte le subroutine finiscono con un RETURN, che segnala all'MSX-BASIC di tornare alla linea successiva al GOSUB che ha chiamato la subroutine. Quando volete entrare in una subroutine, dovete sempre usare un GOSUB, in modo che il RETURN sappia dove deve tornare.
Un buon esempio di una subroutine è il programma che stampa le parole di una canzoncina popolare. Anziché avere molte linee che stampano "la la", potete usare una subroutine.

```
10 REM Uno strano modo per scrivere
20 REM un simpatico ritornello
30 PRINT "Son tre notti che non dormo"
40 GOSUB 140
50 PRINT "ho perduto il mio galletto"
60 GOSUB 140
70 PRINT "poverino"
80 GOSUB 140
90 PRINT "poveretto"
100 GOSUB 140
110 PRINT "l' ho perduto nel giardin
120 GOSUB 140
130 END
140 REM Questa e' la subroutine del la la
150 FOR X = 1 TO 2
160 PRINT "la ";
170 NEXT X
180 PRINT
190 RETURN
```

Quando l'MSX-BASIC incontra il GOSUB alla linea 40, salta alla linea 140 e memorizza che la prossima volta che vedrà un RETURN, dovrà tornare alla linea 50. Nella subroutine, l'MSX-BASIC stampa due volte "la" (l'uso del loop FOR NEXT è, ovviamente, eccessivo, ma vi ricorda l'utilità dei loop). La linea 190 indica la fine della subroutine.
Notate la linea 130: questa volta END non è alla fine del listato. Se ripensate a ciò che sapete sull'esecuzione del programma, vedete che questo è decisamente il miglior posto per mettere l'istruzione END. Se la linea 130 non esistesse, l'MSX-BASIC eseguirebbe la linea 140 dopo essere tornato dall'ultima chiamata di subroutine e all'esecuzione della linea 190 apparirebbe il seguente messaggio di errore:

```
RETURN without GOSUB in 190
```

Questo perché l'MSX-BASIC vuole tornare al programma, ma non sa dove, poiché non siete entrati nella subroutine tramite un GOSUB. Questa è la ragione per cui i programmatori BASIC mettono solitamente un'istruzione END prima dell'inizio di una subroutine. L'MSX-BASIC vi permette di avere nel vostro programma tante istruzioni END quante ne volete.

9.4 Le istruzioni ON GOTO e ON GOSUB

L'istruzione IF THEN è utile per controllare una sola condizione e scegliere uno o due itinerari in base al risultato. Comunque, ci possono essere punti nel programma dove una decisione può avere più di due strade da seguire. Per esempio, potreste chiedere all'utente di battere A, B, C o D. Usare quattro IF THEN è alquanto scòmodo, perciò l'MSX-BASIC ha a disposizione le istruzioni ON GOTO e ON GOSUB.
La sintassi di ON GOTO è

ON *espressione* GOTO *linea* [, *linea*] ...

L'istruzione ON GOSUB ha una sintassi identica. *espressione* è un numero tra 0 e 255. Se il valore di *espressione* è 1, viene eseguita la prima *linea*, se è 2 la seconda e così via. Se la variabile vale 0 o un numero maggiore della lunghezza della lista, l'MSX-BASIC passa alla linea successiva.
Come vedete, usare ON GOTO per riconoscere le scelte dell'utente è molto semplice. Il seguente programma mostra come utilizzare ON GOTO per questo scopo:

```
10 REM Esempio di ON GOTO
20 LINE INPUT "Batti A, B, C o D: "; RISP$
30 ON (ASC(RISP$)-64) GOTO 60, 80, 100, 120
40 PRINT "Non hai battuto A,B,C o D"
50 END
60 PRINT "Hai battuto una A"
70 END
80 PRINT "Hai battuto una B"
90 END
100 PRINT "Hai battuto una C"
110 END
120 PRINT "Hai battuto una D"
130 END
```

L'espressione nell'istruzione ON GOTO usa l'operatore ASC per trovare il codice ASCII della lettera immessa. Poiché il codice ASCII di A è 65, ASC(RISP$)−64, dà 1 per A, 2 per B e così via. Provate a immettere una lettera minuscola per convincervi del risultato in caso di numero più grande della lista di numeri di linea.

9.5 Interruzioni e pause nel programma

Abbiamo visto i vari modi con i quali potete controllare l'esecuzione delle parti di un programma. Inoltre, l'utente dispone di alcuni comandi per controllare direttamente quello che accade nel programma.
Mentre un programma sta girando, l'utente può sospendere l'esecuzione premendo il tasto STOP. Questo dice semplicemente all'MSX-BASIC di non fare nulla finché non viene premuto il tasto STOP un'altra volta. Quindi si deve premere questo tasto per interrompere temporaneamente il programma e poi premerlo di nuovo per continuare l'esecuzione.
Premendo i tasti CTRL e STOP contemporaneamente, uscite dal programma e tornate all'MSX-BASIC. Questo è utile se state sperimentando un programma e ne volete provare alcune parti. La combinazione CTRL STOP è un metodo un po' drastico ma di sicuro successo.

Capitolo

Input da tastiera e stampa del testo 10

Questo capitolo descrive le istruzioni che svolgono i compiti di input e di stampa. Per esempio, avete visto come LINE INPUT riceva il testo dall'utente e come PRINT invii al video il testo. Questi comandi, e quelli ad essi correlati, vengono qui descritti in dettaglio.

10.1 L'acquisizione dell'input da tastiera

Quando scrivete un programma che richiede all'utente di scrivere delle frasi, normalmente fate sì che stampi un messaggio di richiesta e attenda un input da tastiera. L'MSX-BASIC memorizza l'input in una variabile (solitamente una stringa) che potete usare nel resto del vostro programma.

10.2 L'istruzione INPUT

L'istruzione INPUT è il modo più semplice per acquisire caratteri dalla tastiera.
La sintassi è:

INPUT ["*prompt*"]; *variabile*[,*variabile*]

prompt non è indispensabile. Quando l'MSX-BASIC trova un comando

INPUT, stampa un punto di domanda e uno spazio e attende che l'utente batta la risposta e prema il tasto RETURN. Se aggiungete una stringa in *prompt*, l'MSX-BASIC stampa il vostro messaggio prima del punto di domanda. L'utente può usare i tasti INS, DEL e BACKSPACE mentre immette la risposta.
Il caso più semplice di comando INPUT è con un'unica variabile stringa:

```
450 INPUT JX$
```

In questo caso, l'MSX-BASIC legge la riga scritta dall'utente e la pone nella variabile stringa JX$. Tenete presente che:

- L'utente non deve includere virgole nella risposta. Se l'MSX-BASIC vede una virgola, crede che siano state immesse due stringhe invece di una e stampa il messaggio:

```
?Extra ignored
```

  e nella variabile mette solo il testo prima della virgola.
- La stringa non deve iniziare con le virgolette, a meno che l'utente non voglia inserirvi un segno di punteggiatura. Quando l'MSX-BASIC trova le virgolette all'inizio della linea, non le include nella variabile.
- Per includere nel testo una virgola, l'utente deve racchiudere la stringa fra virgolette.

Per inciso, il carattere RETURN non viene posto nella stringa.
Se volete acquisire due stringhe, l'utente deve mettere una virgola tra le variabili, per esempio:

```
450 INPUT JX$,KX$
```

L'utente deve scrivere una stringa, una virgola, e la seconda stringa. Se non viene fornita una seconda stringa, l'MSX-BASIC stampa

```
??
```

Questo è probabilmente il più confuso messaggio di errore che otterrete dall'MSX-BASIC. Esso significa che l'MSX-BASIC è in attesa di ulteriore input.
Se usate INPUT con variabili numeriche, l'utente deve battere un numero. Per esempio se la vostra istruzione INPUT è

```
230 INPUT X
```

e l'utente batte "xxx", l'MSX-BASIC dà il messaggio:

```
?Redo from start
?
```

Anche questo non è un messaggio molto chiaro. È il modo dell'MSX-BASIC per dire all'utente che, qualsiasi cosa sia stata scritta, non era un numero.
Se il vostro INPUT richiede molte variabili, di cui almeno una è numerica, e l'utente batte qualcosa di sbagliato per la variabile numerica, il messaggio appare nuovamente e l'utente deve riscrivere l'intero elenco di variabili.

10.3 L'istruzione LINE INPUT

Come avete visto dalla precedente discussione, INPUT non è il modo migliore per acquisire informazioni dall'utente perché sono molte le occasioni di errore. L'istruzione LINE INPUT è un metodo più pulito ed è questa la ragione per cui è stata usata al posto di INPUT nei programmi dei capitoli precedenti.
La sintassi di LINE INPUT è simile a quella di INPUT:

LINE INPUT ["*prompt*"]; *variabile$*

Notate che LINE INPUT può ricevere una sola variabile e questa deve essere una stringa. Questo rende la vita molto più semplice all'MSX-BASIC, dato che non deve più controllare le virgole o il numero delle variabili. Inoltre LINE INPUT non stampa quel fastidioso punto di domanda.
Il seguente programma vi permette di sperimentare i vari tipi di testo che l'utente può immettere con un'istruzione LINE INPUT. Fatelo girare, e provate molte stringhe diverse, comprendenti quei caratteri che avevano confuso INPUT, come virgole, virgolette, ecc. Quando avete finito, premete il tasto RETURN da solo (cioè immettete una stringa vuota).

```
10 REM Esperimento con LINE INPUT
20 LINE INPUT "Scrivi qualcosa-->"; V$
30 IF V$ = "" THEN END
40 PRINT V$
50 GOTO 20
```

Questo dovrebbe convincervi che LINE INPUT è in grado di acquisire tutto quello che l'utente può scrivere sulla tastiera.

10.4 La funzione INPUT$ e la variabile INKEY$

Vediamo come è possibile ricevere i dati dalla tastiera senza obbligare l'utente a premere il tasto RETURN. Se conoscete l'esatto numero di caratteri che l'utente scriverà, potete usare la funzione INPUT$. L'argomento di INPUT$ è il numero di caratteri che vi aspettate.
Per esempio, la prima versione del programma SI O NO può essere riscritta per accettare un carattere senza obbligare l'utente a premere il tasto RETURN. Potete cambiare la linea 20

```
20 LINE INPUT "Batti S o N: "; RISP$
```

così:

```
20 PRINT "Batti S o N:";
25 RISP$ = INPUT$(1)
```

Quando fate eseguire il programma, notate che la lettera immessa non viene mostrata sullo schermo: questa è un'altra differenza tra la funzione INPUT$ e l'istruzione LINE INPUT. Potete stampare la lettera aggiungendo

```
27 PRINT RISP$;
```

L'ultimo metodo per immettere un testo è usare la variabile INKEY$. Questa variabile è simile alla funzione INPUT$, tranne per il fatto che accetta qualsiasi carattere stia premendo l'utente nell'istante in cui l'MSX-BASIC legge la linea di programma contenente INKEY$. Se l'utente non sta premendo un tasto in quel momento, INKEY$ è una stringa vuota. Inoltre la variabile INKEY$ può contenere un solo carattere.
Usare la variabile INKEY$ è un po' complicato, perché l'utente spesso è indeciso nel premere i tasti. Comunque, provvedete a controllare se INKEY$ è vuota (cioè "nulla"); facilmente l'utente premerà il tasto. Per esempio, potete usare INKEY$ nel programma SI O NO, modificandolo così:

```
20 PRINT "Batti S o N:";
25 IF INKEY$ = "" THEN GOTO 25 ELSE RISP$ = INKEY$
27 PRINT RISP$;
```

La differenza principale tra l'uso della variabile INKEY$ e l'uso della funzione INPUT$ è che nel ciclo di INKEY$ alla linea 25, il cursore non è mostrato sul video. Questo può confondere alcuni utenti abituati al cursore; comunque, se non volete il cursore sullo schermo, usate INKEY$.

L'ACQUISIZIONE DI CARATTERI SPECIALI

Finora avete visto soltanto come immettere caratteri che potete vedere sulla tastiera, vale a dire numeri, lettere e segni di interpunzione. Il vostro computer MSX può stampare molti più caratteri di questi, e tutti si possono immettere dalla tastiera. L'Appendice D mostra tutti i caratteri che potete stampare. Per esempio per stampare una A maiuscola con sopra la dieresi, dovete battere:

```
PRINT CHR$(142)
```

Potreste anche voler acquisire questi caratteri dalla tastiera per mezzo di un'istruzione di input. Per far questo dovete premere sulla vostra tastiera alcuni tasti speciali insieme ai normali tasti.
I tasti speciali sono SHIFT, CODE e GRAPH. Conoscete già il tasto SHIFT: per scrivere le lettere maiuscole, tenete abbassato il tasto SHIFT mentre battete una lettera. I tasti CODE e GRAPH funzionano allo stesso modo, in quanto li dovete tenere abbassati mentre premete un altro tasto.
Per ottenere caratteri speciali, tenete abbassati uno o due tasti speciali. Per esempio, se state scrivendo un testo e volete includervi una A maiuscola con la dieresi, premete i tasti SHIFT e CODE e poi il tasto A. L'Appendice F mostra i tasti che si devono premere per i diversi tipi di caratteri grafici.

10.5 L'istruzione PRINT

Ci sono molti modi per visualizzare un testo sul video del computer MSX, e in tutti è presente l'istruzione PRINT. PRINT è la più versatile istruzione MSX-BASIC e può essere usata per molti tipi di stampa di testi. Avete già visto PRINT in molti dei precedenti programmi.
Tutto quello che si dirà su PRINT, vale anche per l'istruzione LPRINT. Come LLIST manda alla stampante quello che LIST manda sul video, così il comando LPRINT manda alla stampante il testo che il PRINT manda sul video.

STAMPA SEMPLICE

Una forma semplice dell'istruzione PRINT contiene solo una stringa:

PRINT *stringa*

stringa può essere sia una costante, come "Hai battuto S", sia una variabile, come RISP$. In questo caso, la stringa viene stampata e l'MSX-BASIC sposta il cursore all'inizio della linea successiva.

```
10 CLS
20 REM Esempio di PRINT
30 PRINT "Linea 1"
40 PRINT "Linea 2"
50 PRINT
60 PRINT "Linea 4"
```

Potete dare un PRINT senza argomenti se volete solo saltare una riga. Potete stampare più di una sola variabile sulla stessa linea nella stessa istruzione PRINT. Il modo più semplice per farlo è usare il punto e virgola per separare le variabili. Per esempio il seguente programma circonda di asterischi una stringa:

```
10 REM PRINT con ';'
20 PRINT "Batti 2 caratteri:";
30 MM$ = INPUT$(2)
40 PRINT "*****"; MM$; "*****"
```

Qui il punto e virgola viene usato alla linea 20 (per impedire al cursore di spostarsi alla linea seguente) e alla linea 40 (per mettere sulla stessa linea le tre stringhe). Notate che l'MSX-BASIC non mette alcuno spazio tra le stringhe. Se volete inserire degli spazi dovete specificarli nelle stringhe:

```
40 PRINT "*****  "; MM$; "  *****"
```

Allo stesso modo potete stampare dei numeri:

```
10 REM PRINT con ';'
20 PRINT "Batti un numero di 2 cifre:";
30 I1 = VAL(INPUT$(2))
40 PRINT "*****"; I1; "*****"
```

Notate che l'MSX-BASIC mette degli spazi attorno ai numeri; riprovate il programma usando un numero negativo. Come potete vedere, lo spazio davanti al numero è occupato dal segno negativo. Comunque, ci sarà sempre uno spazio dopo il numero.
Se invece dei punti e virgola usate delle virgole tra le variabili, l'MSX-BASIC mette degli spazi tra esse, in modo da incolonnare le variabili nelle colonne 1 e 14 del video. Comunque, questo non funziona sempre allo stesso modo e dipende dal tipo di dati che usate, perciò è più raccomandabile usare i punti e virgola piuttosto che usare le virgole.

Come avete visto alla linea 20 del precedente programma, potete terminare un PRINT con un punto e virgola e l'MSX-BASIC non andrà alla linea seguente. Quindi, le linee

```
150 PRINT "-->";
160 PRINT "<--"
```

stampano

```
--><--
```

Ci sono due funzioni che si possono usare solo come argomento dell'istruzione PRINT; non potete usarle per nessun altro scopo. La prima è la funzione SPC, che agisce come la funzione SPACE$ aggiungendo un certo numero di spazi a una linea. Per esempio,

```
30 PRINT "Questo"; SPC(10); "e quello"
```

stampa

```
Questo          e quello
```

La funzione TAB è molto più utile: si posiziona automaticamente a una certa colonna da voi scelta. La prima colonna dello schermo è la colonna 0, la seguente è la 1 e così via. La funzione TAB rende molto facile la realizzazione di tabelle, poiché solitamente non si sa prima quanto è lungo un numero o una stringa da stampare. Le linee che seguono stampano una lista di nomi con il nome di battesimo allineato alla colonna 13 e l'età allineata alla colonna 20.

```
320 PRINT "Cognome"; TAB(14); "Nome"; TAB(21); "Eta'"
330 PRINT C1$; TAB(14); N1$; TAB(21); E1
340 PRINT C2$; TAB(14); N2$; TAB(21); E2
350 PRINT C3$; TAB(14); N3$; TAB(21); E3
```

Finché ogni nome rientra nella sua colonna, tutto viene allineato. Se date una funzione TAB con un argomento che sposta in cursore indietro, l'MSX-BASIC la ignora. Quindi se C2$ fosse più lunga di 14 caratteri, N2$ sarebbe stampata sopra di essa. Per evitare questo, potete usare la funzione LEFT$ per eliminare tutto quello che contrasta con la funzione TAB.

```
320 PRINT "Cognome"; TAB(14); "Nome"; TAB(21); "Eta'"
330 PRINT LEFT$(C1$, 12); TAB(14); LEFT$(N1, 6);
TAB(21); E1
340 PRINT LEFT$(C2$, 12); TAB(14); LEFT$(N2, 6);
TAB(21); E2
```

```
350 PRINT LEFT$(C3$, 12); TAB(14); LEFT$(N3, 6);
TAB(21); E3
```

STAMPA FORMATTATA

Stampare con PRINT e con le molte funzioni stringa vi permette una grande versatilità, ma c'è un'altra forma dell'istruzione PRINT ancora più potente: l'istruzione PRINT USING, che vi permette di specificare il formato della linea di stampa per mezzo di una stringa di caratteri speciali. La sintassi dell'istruzione è:

PRINT USING "*stringaformato*"; *argomento1*, *argomento2*, ...

dove *stringaformato* è una stringa, costante o variabile, e gli *argomenti* possono essere sia costanti che variabili.
Sfortunatamente preparare *stringaformato* è spesso un procedimento difficile: se volete usare un formato semplice di stampa è meglio che usiate la normale istruzione PRINT.
I caratteri speciali che possono essere usati in una stringa di formato, sono mostrati nelle Figure 10.1 e 10.2. In una stringa di formato ogni carattere che non si trova in queste liste viene trattato come un normale carattere e stampato esattamente come appare.
Se tutto questo vi sembra un po' confuso, non arrendetevi. Il primo esempio, che stampa delle cifre, può schiarirvi un po' le idee. In una stringa di formato, si utilizza il simbolo di numero (#) per indicare il posto dove può stare una cifra. Fate girare il seguente programma:

```
10 REM Esempio di PRINT USING
20 N1 = 1234
30 PRINT USING "ooo####ooo"; N1
```

La stampa di questo programma è

```
ooo1234ooo
```

Le "o" dimostrano che l'MSX-BASIC stampa direttamente tutti i caratteri non di formato che si trovano nella stringa di formato. Siccome avete indicato quattro spazi per numeri, l'MSX-BASIC ha stampato quattro cifre del numero.
Ora modificate leggermente il programma:

```
10 REM Esempio di PRINT USING
20 N1 = 1234
30 PRINT USING "ooo####ooo"; N1
40 PRINT USING "ooo#######ooo"; N1
```

Carattere	Significato
#	Spazio per una cifra
.	Include un punto decimale
+	Indica + o - ; può essere usato prima o dopo il numero
-	Indica solo - ; può essere usato dopo il numero
££	Pone un £ alla sinistra del numero
**	Pone tanti * alla sinistra del numero per l'intera lunghezza del campo
**£	Pone un £ preceduto da tanti * per l'intera lunghezza del campo
^^^^	Visualizza il numero in notazione scientifica

Figura 10.1 Caratteri di formato per dati numerici

Carattere	Significato
\ \	Spazio per molti caratteri
!	Spazio per un carattere
-	Successivo carattere alfabetico
altro	Lo invia in output

Figura 10.2 Caratteri di formato per testi

Ci sono sette spazi per numeri, sebbene 1234 sia solo di quattro cifre. Il risultato è

```
ooo1234ooo
ooo    1234ooo
```

Quando usate il simbolo di numero, l'MSX-BASIC lascia sempre tanti posti quanti ne avete specificati. Se il numero stampato occupa meno posti

di quelli che avete specificato, l'MSX-BASIC riempirà i posti in più con degli spazi sulla sinistra.
Questa formattazione viene chiamata giustificazione a destra ed è molto utile quando avete una colonna di numeri che volete addizionare. Confrontate l'output delle due parti del programma che segue:

```
10 REM Addizione incolonnata con il vecchio metodo
20 X1 = 6391
30 X2 = 934
40 X3 = 5197
50 PRINT X1
60 PRINT X2
70 PRINT X3
80 PRINT " -----"
90 PRINT X1+X2+X3
92 PRINT:PRINT
100 REM Addizione incolonnata con il nuovo metodo
110 PRINT USING "#####"; X1
120 PRINT USING "#####"; X2
130 PRINT USING "#####"; X3
140 PRINT "-----"
150 PRINT USING "#####"; X1+X2+X3
```

il risultato è

```
 6391
 934
 5197
 -----
 12522

 6391
  934
 5197
-----
12522
```

Notate come è difficile addizionare mentalmente la prima serie di numeri, poiché non sono incolonnati nel modo che siete abituati ad usare. La seconda serie di numeri è molto più chiara.
Se state usando numeri reali, potete usare un punto nella stringa di formato, per specificare dove volete che appaia il punto decimale. L'MSX-BASIC arrotonda i decimali in più o aggiunge zero se ci sono troppo pochi decimali:

```
10 REM PRINT USING con '.'
20 PRINT USING "###.###"; 123.456
30 PRINT USING "###.###"; 1.23456
```

```
40 PRINT USING "###.###"; 1
50 PRINT USING "###.###"; 1.1
```

La stampa è

```
123.456
  1.235
  1.000
  1.100
```

Notate come 1.23456 sia stato arrotondato a 1.235 e non troncato a 1.234. L'MSX-BASIC usa le regole standard di arrotondamento.
Potete usare il segno più (+) prima o dopo il numero per stampare il segno prima o dopo il numero. Potete usare anche un segno meno dopo il numero per stampare un "−" dopo di esso se è negativo.

```
10 REM PRINT USING con '+' e '-'
20 PRINT USING "+####"; 721
30 PRINT USING "+####"; -721
40 PRINT USING "####+"; 93
50 PRINT USING "####+"; -93
60 PRINT USING "####-"; 128
70 PRINT USING "####-"; -128
```

Il risultato è

```
+721
-721
 93+
 93-
128
128-
```

Notate che il numero stampato occupa tanti posti quanti sono indicati dai simboli di numero, più uno per l'eventuale segno.
Mettendo due simboli di lira (£) — nel Sony si ottiene £ premendo SHIFT, CODE e 4 — a sinistra dei simboli di numero, l'MSX-BASIC trasporterà il simbolo di lira direttamente alla sinistra del numero stampato. Per esempio

```
210 PRINT USING "££######"; 7732
220 PRINT USING "££######"; 812
```

stampa

```
£7732
 £812
```

Avrete senz'altro visto degli assegni scritti con asterischi prima del numero per evitare le contraffazioni. L'istruzione PRINT USING vi permette di far precedere il numero da asterischi mettendo due asterischi nella stringa di formato. Per esempio:

```
210 PRINT USING "******.##"; 7732
```

stampa:

```
****7732
```

Se volete sia gli asterischi che il simbolo di lira, scrivete "**£" nella stringa di formato:

```
210 PRINT USING "**£#######";7732
****£7732
```

È anche possibile separare le migliaia nei numeri stampati; basta aggiungere una virgola a sinistra del punto nella stringa di formato. Ogni virgola stampata aggiunge un posto alla stringa di stampa.

```
80 PRINT USING "########,.##"; 6328900.99#

6,328,900.99
```

Potete usare quattro segni (^^^^) per indicare che il numero deve essere stampato in notazione scientifica. Usate i simboli di numero come avete fatto prima per indicare le cifre a sinistra e a destra del punto decimale. Per esempio,

```
320 PRINT USING "#.####^^^^"; 35.7182
```

stampa

```
0.3572E+02
```

Con l'istruzione PRINT USING potete anche decidere il formato di una stringa. Per specificare quanti caratteri di una stringa volete stampare, usate due *backslash* (\) e spazi vuoti. Per esempio, per specificare di lasciare cinque posti, usate "backslash, 3 spazi e backslash", come nel seguente programma:

```
10 REM PRINT USING con '\'
20 S1$ = "mnopqrst"
30 S2$ = "xx"
```

```
40 PRINT USING "yyy\    \yyy"; S1$
50 PRINT USING "yyy\    \yyy"; S2$
```

Il risultato è

```
yyymnopqyyy
yyyxx   yyy
```

Se la stringa è troppo lunga, come S1$, vengono usati i caratteri più a sinistra. Se la stringa è troppo corta, come S2$, vengono messi degli spazi a destra.
Il punto esclamativo (!) indica che deve essere stampato solo il primo carattere della stringa. Il programma

```
10 REM PRINT USING con '!'
20 S1$ = "mnopqrst"
30 S2$ = "xx"
40 PRINT USING "yyy!yyy"; S1$
50 PRINT USING "yyy!yyy"; S2$
```

stampa

```
yyymyyy
yyyxyyy
```

Una "&" concatena la stringa che la segue alla stringa da stampare.

10.6 Come stampare in qualunque punto dello schermo

L'istruzione PRINT inizia a stampare nel punto in cui si trova il cursore quando voi date il comando. Se terminate un PRINT con un punto e virgola, il cursore rimane nella posizione dopo l'ultimo carattere stampato. Potreste voler spostare il cursore in differenti parti dello schermo, prima di stampare del testo e l'MSX-BASIC ve ne dà la possibilità.
Nel modo testo, lo schermo ha 24 righe in verticale e 39 colonne in orizzontale. L'MSX-BASIC usa le coordinate cartesiane per individuare ogni posizione sullo schermo. Ogni posizione ha una coordinata x e una coordinata y, che vengono indicate come (x, y).
Le coordinate x vanno da 0 a 39, da sinistra a destra, e le coordinate y vanno da 0 a 23, dall'alto in basso (considerando che la linea che indica i tasti funzione sia disattivata, come vedremo). Quindi l'angolo in alto a sinistra si indica con (0,0) e quello in basso con (38,23). La Figura 10.3 mo-

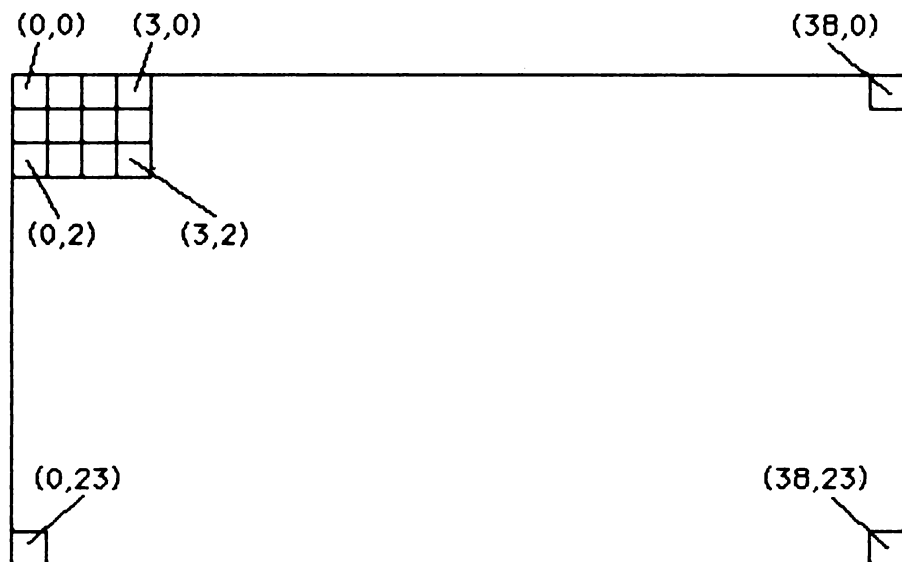



Figura 10.3 Coordinate dello schermo in modo testo

stra una rappresentazione dello schermo con l'indicazione di alcune coordinate.
La prima cosa che dovete fare è eliminare la linea dei tasti funzione sul fondo dello schermo e poi pulire lo schermo. L'istruzione KEY OFF cancella la linea dei tasti funzione. L'istruzione CLS è utile insieme all'istruzione PRINT, poiché ripulisce dal testo lo schermo. Dopo un CLS, il cursore si trova nell'angolo in alto a sinistra dello schermo in (0,0).

```
10 REM Posizionamento del testo
20 KEY OFF
40 CLS
```

Il comando LOCATE posiziona il cursore in un punto specifico. La sua sintassi è

LOCATE *x*, *y*

Fate girare il seguente programma:

```
10 REM Posizionamento del testo
```

```
20 KEY OFF
30 CLS
40 LOCATE 19, 12
50 PRINT "*";
60 TIME = 0
70 IF TIME < 100 THEN GOTO 70
```

Questo programma colloca un asterisco nel mezzo dello schermo e attende due secondi. Questo invece disegna una linea di asterischi dall'angolo in basso a sinistra al centro del margine superiore.

```
10 REM Disegna una diagonale
20 KEY OFF
30 CLS
40 FOR I = 22 TO 0 STEP -1
50 LOCATE (22-I), I
60 PRINT "*"
70 NEXT I
```

L'MSX-BASIC vi permette di tralasciare anche uno dei valori dell'istruzione LOCATE, purché scriviate sempre la virgola. Se tralasciate uno degli argomenti, l'MSX-BASIC assume che volete che il cursore resti nella stessa colonna o nella stessa linea in cui era prima. Per esempio, il programma che segue stampa cinque asterischi in verticale iniziando dalla colonna in cui si trova il cursore:

```
150 CLS
160 FOR N = 0 TO 4
170 LOCATE , N
180 PRINT "*"
190 NEXT N
```

Mentre scrivete un programma, non potete essere sicuri di dove si trovi il cursore sullo schermo. La funzione POS vi dà il valore della coordinata *x*. L'argomento della funzione POS può essere qualsiasi numero, dato che l'MSX-BASIC ignora l'argomento. La variabile CRSLIN contiene invece il valore della coordinata *y*.
Per esempio le seguenti linee memorizzano nelle variabili X9 e Y9 la posizione attuale del cursore:

```
410 X9 = POS(1)
420 Y9 = CRSLIN
```

Se state usando l'istruzione LPRINT per mandare il testo su una stampante, potete usare la funzione LPOS nello stesso modo della funzione POS. Per la stampante non esiste un equivalente della variabile CRSLIN.

Capitolo

La grafica

Quasi tutti i programmi che abbiamo visto finora, manipolano solamente del testo. Se avete comprato una qualsiasi cartuccia di giochi per il vostro computer, avrete visto sullo schermo ogni genere di colori e forme; tutte le caratteristiche che vedete nei giochi, si possono comunque realizzare con l'MSX-BASIC.
La grafica al computer è, nella moderna tecnologia, un campo molto interessante e trova applicazione nella pubblicità e in molti film. Un'immagine creata con il computer ha tre caratteristiche fondamentali: il colore, il movimento (l'animazione) e la forma. I disegni che otterrete con il vostro calcolatore non saranno così spettacolari come quelli della televisione, ma vi daranno un'idea di come vengono create le immagini col computer.
L'MSX-BASIC ha una grande versatilità nel disegnare immagini sul video. Per esempio, potete scegliere fino a quindici colori diversi e programmare gli sprite, che sono piccoli disegni mobili utilizzabili per i giochi.
Il computer MSX ha quattro modi di visualizzazione.

- Modo 0 — Testo standard. Questo è il modo con il quale abbiamo lavorato finora che non permette alcuna visualizzazione grafica. Può essere visualizzato solo il testo in una matrice 40×24, con la scelta di un colore per i caratteri (detto colore di primo piano) e di un colore per lo sfondo.
- Modo 1 — Testo multicolore. Come il modo 0, visualizza principalmente testi, ma ogni serie di otto lettere può avere un suo colore di primo piano e di sfondo. In questo modo potete usare sprite (ma non disegni) e potete visualizzare testi in una matrice 32×40.

- Modo 2 — Grafica ad alta risoluzione. È il modo usato dalla maggior parte dei giochi. Potete mettere dei puntini (detti *pixel*) dove volete in una matrice 256×192 e usare gli sprite. Ogni serie di otto pixel ha i propri colori di primo piano e di sfondo. Buona parte degli esempi presentati in questo capitolo viene visualizzata in questo modo.
- Modo 3 — Grafica a bassa risoluzione. La matrice in questo modo è solamente 64×48, il che lo rende inutile per molte applicazioni. Comunque, ogni pixel ha il proprio colore e potete usare gli sprite.

Avete già visto parte di quello che potete fare nel modo 2, nel programma PRIMO; in questo capitolo tratteremo tutti gli aspetti della grafica disponibile con l'MSX-BASIC.

11.1 Il colore

Le solite lettere bianche su sfondo nero possono avervi un po' stufato, ormai. Dopotutto "il mondo è un carosello di colori", ma l'MSX non lo ha ancora dimostrato. Nella programmazione grafica, è molto semplice controllare il colore con l'istruzione COLOR. La sintassi di COLOR è:

COLOR *primopiano, sfondo, bordo*

Ognuno degli argomenti deve essere compreso da 0 a 15. La Figura 11.1 elenca i colori disponibili per un normale televisore. Il televisore è in genere più luminoso di un normale monitor per computer: è quindi consigliabile diminuire la luminosità e aumentare il contrasto. Se fate ciò la saturazione dei colori può variare in modo notevole; in altre parole, potreste aver bisogno di stilare una vostra tabella dei colori facendo degli esperimenti.

Nel modo 0, quello in cui si scrivono i programmi, non c'è colore del bordo. In questo modo avete 256 possibilità per l'abbinamento primo piano/sfondo; molti di questi accoppiamenti non sono interessanti perché i colori sono troppo simili, il che rende il testo illeggibile, o perché i colori stonano fra loro. Il seguente programma vi dà alcuni esempi di coppie di colori.

```
10 REM Esperimenti di abbinamento dei colori
20 CLS
30 PRINT "Bianco su blu (15,4)"
40 GOSUB 1000
50 COLOR 10,6
60 PRINT "Giallo scuro su rosso (10,6)"
70 GOSUB 1000
```

```
80 COLOR 1,15
90 PRINT "Nero su bianco (1,15)"
100 GOSUB 1000
110 COLOR 4,15
120 PRINT "Blu su bianco (4,15)"
130 GOSUB 1000
140 COLOR 12,8
150 PRINT "Verde scuro su rosso (12,8)"
160 GOSUB 1000
170 COLOR 10,6
180 END
1000 REM Subroutine di attesa (3 sec)
1010 TIME = 0
1020 IF TIME < 150 THEN GOTO 1020
1030 RETURN
```

Numero	Colore
1	Nero
2	Verde
3	Verde chiaro
4	Blu
5	Azzurro
6	Rosso mattone
7	Turchese
8	Rosso arancio
9	Arancio
10	Giallo scuro
11	Giallo
12	Verde scuro
13	Lavanda
14	Grigio
15	Bianco
0	Trasparente (l'oggetto assume il colore dello sfondo)

Figura 11.1 Colori disponbili in MSX-BASIC

Questo programma invece passa in rassegna tutte le possibilità di abbinamento dei colori. Naturalmente quando lo sfondo e il primo piano sono uguali, il testo è illeggibile.

```
10 REM Dimostrazione dei colori
20 KEY OFF
30 CLS
40 FOR N = 1 TO 23
50 PRINT STRING$(37,"X");
60 NEXT N
70 REM S = sfondo, P = primo piano
80 LOCATE 0,0
90 PRINT "P=  "
100 PRINT "S=  "
110 REM Visualizza tutti i colori di primo piano
111 REM per ogni sfondo
120 FOR S = 1 TO 15
130 LOCATE 2,1
140 PRINT USING "##"; S
150 FOR P = 1 TO 15
160 LOCATE 2,0
170 PRINT USING "##"; P
180 COLOR P, S
190 TIME = 0
200 IF TIME < 75 THEN GOTO 200
210 NEXT P
220 NEXT S
230 REM Fine
240 CLS
250 COLOR 15,4
```

11.2 Grafica in modo 2

Ora che avete visto i colori che l'MSX-BASIC vi mette a disposizione, potete iniziare ad usare la grafica nel modo 2. La prima istruzione di cui avete bisogno, è SCREEN che seleziona il modo di visualizzazione.
La sua sintassi è

SCREEN *modo*

dove *modo* è il numero del modo in cui volete entrare.
Nel modo 2, l'istruzione COLOR predispone il valore del corrente colore di primo piano e dello sfondo. Potete avere nello stesso momento molti colori di primo piano sullo schermo, come avete visto con i rettangoli colorati nel programma PRIMO.
La maggior parte dei comandi per disegnare sullo schermo nel modo 2, ha la possibilità di scegliere il colore principale per la durata del comando; questo comunque, non diventa il colore corrente di primo piano. Se in un comando non specificate il colore, viene utilizzato quello di primo piano.

Dopo aver usato nel modo 2 COLOR per selezionare il colore dello sfondo, dovete usare CLS per avere quel colore sullo schermo. Questa è la ragione per cui vedete spesso nei programmi delle linee come

```
210 COLOR ,1 : CLS
```

Il modo 2 non ha un corrispondente della funzione POS e della variabile CRSLIN: se vi dimenticate la posizione del cursore grafico, non c'è nessun comando o funzione che ve la può rilevare direttamente. L'MSX-BASIC ha a disposizione la funzione POINT per dirvi di che colore è un pixel. La sintassi della funzione POINT è:

POINT (*x*, *y*)

Come è facile immaginare, il suo valore varia da 0 a 15, a seconda del colore del punto (*x*, *y*).
Quando programmate, ricordate che ogni serie di otto pixel in orizzontale può avere un suo colore. Questo restringe purtroppo le vostre possibilità, poiché potete cambiare colore solamente dopo ogni otto pixel. Comunque, buona parte dei comandi che si trovano in questo capitolo, funzionano allo stesso modo sia nel modo 3 sia nel modo 2.

DISEGNARE PUNTI

Ricordate che il modo 2 ha una matrice di 256×192 pixel. Questi vengono individuati per mezzo delle coordinate cartesiane, a partire da (0,0) nell'angolo in alto a sinistra. La Figura 11.2 mostra come è suddiviso lo schermo in questo modo.
È importante sottolineare che molti televisori, usati come monitor per il computer, non mostrano tutti i pixel ai margini sinistro o destro, a seconda della loro predisposizione: molti non mostrano i pixel delle colonne dalla 0 alla 8.
Invece dell'istruzione LOCATE, che avete usato per muovere il cursore del testo sullo schermo, nel modo 2 dovete usare PSET per posizionare il cursore grafico. Notate, comunque, che nel modo 2 tutte le istruzioni per disegnare sullo schermo aggiornano la posizione del cursore grafico. L'istruzione PSET non solo aggiorna la posizione del cursore grafico, ma disegna anche un pixel in quella posizione. La sintassi dell'istruzione PSET è:

PSET [STEP] (*x*, *y*) [,*colore*]

Gli argomenti STEP e *colore* non sono indispensabili e per questo sono stati racchiusi tra parentesi quadre.

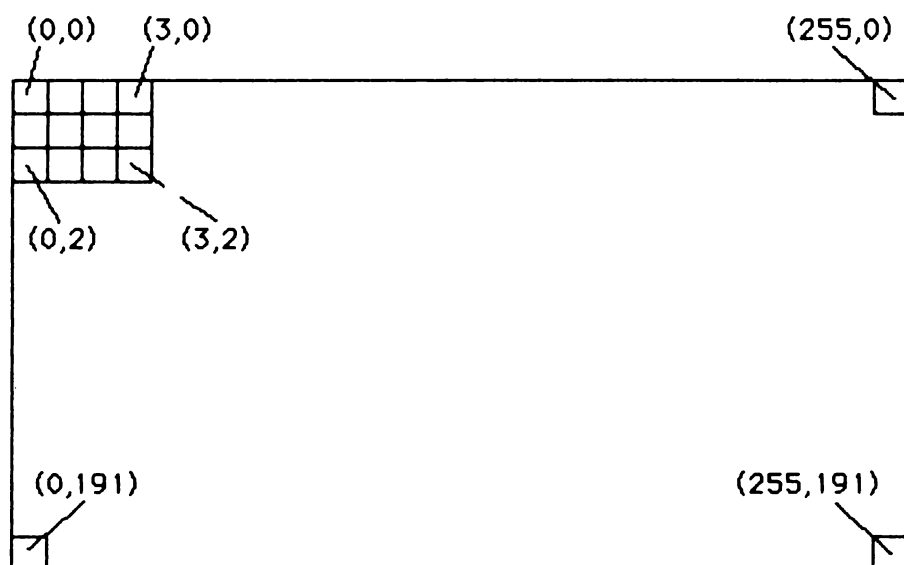



Figura 11.2 Coordinate dello schermo in modo grafico

L'esempio che segue, colora lo sfondo di rosso (colore 6) e disegna un quadrato azzurro con gli angoli nei punti (100,100), (100,150), (150,150) e (150,100).

```
10 REM Uso di PSET
20 SCREEN 2
30 COLOR ,6 : CLS
40 FOR Y = 100 TO 150
50 PSET (100, Y), 5
60 NEXT Y
70 FOR X = 100 TO 150
80 PSET (X, 150),5
90 NEXT X
100 FOR Y = 150 TO 100 STEP -1
110 PSET (150, Y), 5
120 NEXT Y
130 FOR X = 150 TO 100 STEP -1
140 PSET (X, 100), 5
150 NEXT X
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

L'argomento STEP dell'istruzione PSET (e delle altre istruzioni grafiche che vedrete) semplifica molto il disegno. Finora, avete specificato l'esatta posizione di ogni pixel. Con l'argomento STEP, i valori *x* e *y* non esprimono la posizione del pixel, ma la posizione relativa al cursore grafico. Per esempio, per disegnare un punto 30 pixel a destra e 15 pixel sotto il cursore grafico, indipendentemente dalla sua attuale posizione, date il comando:

```
PSET STEP (30,15)
```

Per disegnare un pixel a sinistra e due pixel sopra il cursore grafico, date il comando

```
PSET STEP (-1,-2)
```

Potete vedere come questo semplifica il programma che disegna il quadrato.

```
10 REM Uso di PSET e di STEP
20 SCREEN 2
30 COLOR ,6 : CLS
35 PSET (100, 100), 5
40 FOR I = 1 TO 50
50 PSET STEP (0, 1), 5
60 NEXT I
70 FOR I = 1 TO 50
80 PSET STEP (1, 0), 5
90 NEXT I
100 FOR I = 1 TO 50
110 PSET STEP (0, -1), 5
120 NEXT I
130 FOR I = 1 TO 50
140 PSET STEP (-1, 0), 5
150 NEXT I
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

In questo caso si aggiunge semplicemente la linea 35 per designare il punto di partenza; poi si usano i loop FOR NEXT per disegnare i punti uno ad uno.
Il vantaggio di usare l'argomento STEP consiste nel poter spostare l'intero disegno cambiando solamente la posizione del primo pixel. Se decidete di disegnare il quadrato partendo da (37, 91), dovete solo modificare la linea 35 in

```
35 PSET (37,91), 5
```

Ricordate che potete scegliere il colore corrente di primo piano con COLOR. Potete ulteriormente semplificare il programma predisponendo il colore con cui disegnare alla linea 30 e non ribadirlo in nessun altro punto:

```
10 REM Uso di PSET e di STEP
20 SCREEN 2
30 COLOR 5,6 : CLS
35 PSET (100, 100), 5
40 FOR I = 1 TO 50
50 PSET STEP (0, 1), 5
60 NEXT I
70 FOR I = 1 TO 50
80 PSET STEP (1, 0), 5
90 NEXT I
100 FOR I = 1 TO 50
110 PSET STEP (0, -1), 5
120 NEXT I
130 FOR I = 1 TO 50
140 PSET STEP (-1, 0), 5
150 NEXT I
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

L'istruzione PRESET è simile a PSET e la sua sintassi è la stessa:

PRESET [STEP] (*x*, *y*) [,*colore*]

Gli argomenti STEP e *colore* sono opzionali. La differenza fra le due istruzioni è che, se non date l'argomento *colore*, PRESET disegna nel colore dello sfondo anziché in quello corrente di primo piano, in pratica cancella invece di disegnare.
Se indicate la scelta di *colore*, PRESET funziona come PSET. Pensandoci bene, vi accorgerete che l'istruzione PRESET è come PSET con l'opzione *colore* per lo sfondo. PRESET è utile solamente se non sapete quale sia il colore dello sfondo.

DISEGNARE LINEE RETTE E QUADRATI

Disegnare linee rette è una delle procedure più comuni nella scrittura di programmi di grafica e può essere noioso scrivere loop FOR NEXT ogni volta che si vuole disegnare una linea. Inoltre, è semplice disegnare linee parallele ai bordi dello schermo: immaginatevi però il loop FOR NEXT che dovreste scrivere per disegnare una linea tra (100, 100) e (141, 107). La funzione matematica diventerebbe alquanto complicata.

Per rendere tutto ciò più facile, l'MSX-BASIC ha l'istruzione LINE. La sintassi dell'istruzione LINE è piuttosto complessa; vediamola passo per passo. In una forma simile a PSET, la sintassi è:

LINE (*x1, y1*)−(*x2, y2*) [,*colore*]

Ancora una volta l'argomento *colore* non è indispensabile. Come potete immaginare, questa istruzione disegna una linea da (*x1, y1*) a (*x2, y2*); alla fine dell'esecuzione dell'istruzione, il cursore grafico si trova in (*x2, y2*). Con l'istruzione LINE possiamo semplificare e accorciare il programma che disegna i quadrati:

```
10 REM Uso di LINE
20 SCREEN 2
30 COLOR 5,6 : CLS
40 LINE (100, 100) - (100, 150)
50 LINE (100, 150) - (150, 150)
60 LINE (150, 150) - (150, 100)
70 LINE (150, 100) - (100, 100)
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Per disegnare molte linee di seguito, una attaccata all'altra, l'MSX-BASIC può utilizzare una versione dell'istruzione LINE ancora più compatta:

LINE−(*x2, y2*) [,*colore*]

In questa forma LINE inizia a tracciare la linea dalla posizione del cursore grafico, che è stata aggiornata dal precedente comando di disegno. Questo è, naturalmente, più semplice che indicare dove si trova il cursore grafico ogni volta. Il programma può essere ulteriormente semplificato:

```
10 REM Uso di LINE
20 SCREEN 2
30 COLOR 5,6 : CLS
40 LINE (100, 100) - (100, 150)
50 LINE - (150, 150)
60 LINE - (150, 100)
70 LINE - (100, 100)
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

L'argomento STEP rende questa struttura più elegante:

LINE−STEP (*x2, y2*) [,*colore*]

Come nell'istruzione PSET, l'argomento STEP rende il vostro programma più flessibile. Il programma che disegna quadrati ora diventa:

```
10 REM Uso di LINE e STEP
20 SCREEN 2
30 COLOR 5,6 : CLS
35 PSET (100, 100)
40 LINE - STEP (0, 50)
50 LINE - STEP (50, 0)
60 LINE - STEP (0, -50)
70 LINE - STEP (-50, 0)
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Ancora una volta, per cambiare la posizione del quadrato dovete solo modificare la linea 35.
Una forma meno usata dell'istruzione LINE è:

LINE STEP (*x1, y1*)–STEP (*x2, y2*) [,*colore*]

Vediamo ora come disegnare dei quadrati: anche questa è un'operazione che vi capiterà di compiere spesso nella programmazione grafica. Se volete disegnare un quadrato con i lati paralleli ai bordi dello schermo, l'MSX-BASIC vi rende la vita semplice: basta aggiungere l'argomento B nell'istruzione LINE. L'argomento B (che sta per *box*), segue l'argomento *colore* e fa disegnare all'MSX-BASIC un quadrato i cui opposti vertici si trovano in (*x1, y1*) e (*x2, y2*).
L'argomento B semplifica ancora il nostro programma esempio.

```
10 REM Uso di LINE e B
20 SCREEN 2
30 COLOR 5,6 : CLS
40 LINE (100, 100) - (150, 150), , B
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Se non specificate il *colore*, dovete comunque mettere la virgola ad esso relativa, in modo che il computer non tenti di colorare la linea con il valore della variabile chiamata B. Potete usare l'argomento B anche insieme all'argomento STEP:

```
10 REM Uso di LINE,B e STEP
20 SCREEN 2
30 COLOR 5,6 : CLS
35 PSET (100, 100)
40 LINE - STEP (50, 50), , B
```

```
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

La linea 40 da sola contiene tutto quello che volete fare, cioè disegnare un quadrato 50×50.
Per disegnare un parallelogramma, potete modificare il programma in:

```
10 REM Parallelogramma
20 SCREEN 2
30 COLOR 5,6 : CLS
35 PSET (100, 100)
40 LINE - STEP (0, 50)
50 LINE - STEP (20,30)
60 LINE - STEP (0, -50)
70 LINE - STEP (-20, -30)
160 TIME = 0
170 IF TIME < 250 THEN GOTO 170
```

Naturalmente, potete disegnare qualsiasi figura. Il programma seguente disegna cento linee a caso collegate tra di loro su tutto lo schermo:

```
10 REM Disegno di linee
20 SCREEN 2
30 COLOR 5,6 : CLS
40 A = RND(-TIME)
50 PSET (0, 0)
60 FOR I = 1 TO 50
70 LINE - ((RND(1)*255), (RND(1)*191))
80 NEXT I
90 TIME = 0
100 IF TIME < 250 THEN GOTO 100
```

DISEGNARE LINEE CURVE E CERCHI

Per produrre disegni più complessi e realistici dobbiamo imparare a disegnare linee curve.
Anche in questo caso l'MSX-BASIC ha un'istruzione apposita: CIRCLE. Malgrado il suo nome, CIRCLE vi permette di disegnare una quantità di forme: cerchi, ellissi, archi di circonferenza e di ellissi. Come l'istruzione LINE, CIRCLE ha una sintassi complessa. La forma più semplice disegna un cerchio:

CIRCLE [STEP] (*x*, *y*), *raggio* [,*colore*]

Al solito, gli argomenti STEP e *colore* si possono anche omettere. L'MSX-

BASIC disegna un cerchio, il cui centro è il punto (*x*, *y*), con il raggio stabilito. Se usate l'argomento STEP, il centro è relativo all'attuale posizione del cursore grafico. Dopo l'esecuzione dell'istruzione CIRCLE, il cursore grafico si trova a sinistra del centro.
Il programma seguente disegna sullo schermo venticinque cerchi di raggio 10:

```
10 REM Cerchi
20 SCREEN 2
30 COLOR 1,6 : CLS
40 PRESET (68, 46)
50 FOR A = 1 TO 5
60 FOR B = 1 TO 5
70 CIRCLE STEP (20, 0), 10
80 NEXT B
90 PRESET STEP (-100, 20)
100 NEXT A
110 TIME = 0
120 IF TIME < 500 THEN GOTO 120
```

A proposito di questo programma, ci sono da notare alcune cose:

- I cerchi vengono tracciati molto velocemente.
- I cerchi non sono perfettamente rotondi: i contorni sono irregolari. Ciò è dovuto al fatto che i pixel hanno una ben definita dimensione. Disegnare una curva con dei pixel produce sempre le dentellature che vedete.
- Quando i cerchi si toccano, le dentellature si notano di più. Se state usando un televisore, noterete che i punti in cui i cerchi si toccano non sono tutti uguali; spostandosi lungo lo schermo i punti di contatto appaiono diversi. Ciò dipende dal modo in cui il televisore mostra sul video i differenti punti colorati. Provate a cambiare i colori alla linea 30 e notate come cambiano questi punti colorati.
- Il programma usa PRESET invece di PSET in modo che non venga disegnato il punto. L'istruzione PRESET viene qui usata semplicemente per spostare il cursore grafico e non per disegnare.

L'istruzione CIRCLE vi permette di disegnare archi aggiungendo due argomenti dopo il colore:

CIRCLE STEP (*x*, *y*), *raggio*, *colore*, *angoloiniziale*, *angolofinale*

Gli argomenti *angoloiniziale* e *angolofinale* sono misurati in radianti. Vediamo brevemente cosa significa misurare gli angoli in radianti.
Probabilmente ricordate che un cerchio è diviso in 360 gradi: 0 gradi è il punto sul cerchio direttamente a destra del centro, 90 gradi è sopra, 180 a sinistra e 270 sotto.

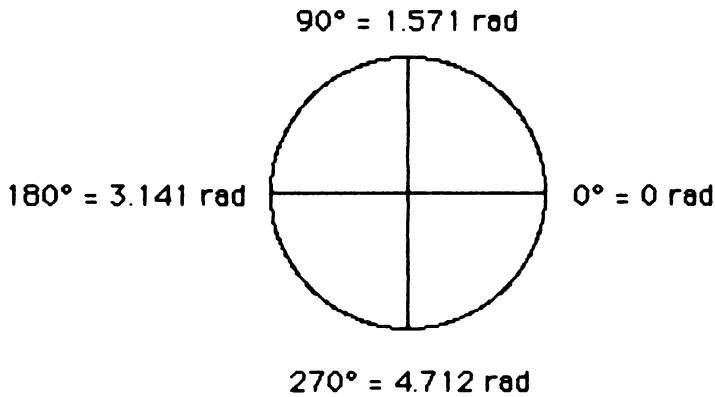



Figura 11.3 Misurazione degli angoli in radianti

In un cerchio ci sono 2*PI radianti (PI è 3.14159). Ciò significa che 360 gradi corrispondono a 6.28318 radianti, ovvero:

1 radiante = 57.2958 gradi
1 grado = 0.01745 radianti

La Figura 11.3 mostra un cerchio con le relative misure sia in gradi che in radianti. Per disegnare un arco, dovete conoscerne l'inizio e la fine in radianti. Siccome la maggior parte delle persone pensa in termini di gradi, anziché in radianti, è comodo avere una costante di conversione nei programmi. Per esempio, per disegnare un arco da 45 gradi (posizione della lancetta sulle ore 1:30) a 135 gradi, il cui centro sia in (80, 60) e che abbia un raggio di 25, potete usare le seguenti istruzioni:

```
RC = 0.01745
CIRCLE (80,60), 25, , 45*RC, 135*RC
```

Il risultato è mostrato nella Figura 11.4 a sinistra.
L'ordine degli angoli di inizio e di fine dell'arco è importante: a destra nella figura vedete il risultato con gli angoli di inizio di inizio e di fine invertiti:

```
CIRCLE (80,60), 25, , 135*RC, 45*RC
```

Un'ellisse è un cerchio allungato in una direzione. Immaginate un qua-

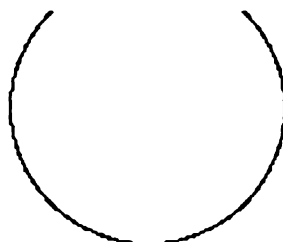
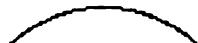



Figura 11.4 Disegno di archi

drato circoscritto ad un cerchio: se allungate il quadrato in un rettangolo il cerchio si allungherà in un'ellisse. L'istruzione CIRCLE vi permette di disegnare un'ellisse specificando il rapporto delle dimensioni del rettangolo, detto rapporto di allungamento.
Per l'ellisse, la sintassi dell'istruzione CIRCLE è:

CIRCLE STEP (*x*, *y*), *raggio*, *colore*, [*angoloiniziale*, *angolofinale*], *rapporto*

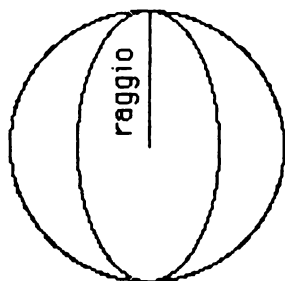
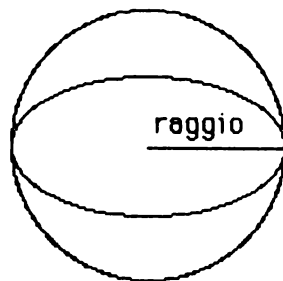



Figura 11.5 Rapporti di allungamento

Se indicate i due angoli, il computer disegnerà un arco di ellisse. Se il rapporto di allungamento è maggiore di uno, l'ellisse è alta e stretta, se è minore di uno, sarà bassa e larga.
Il raggio è sempre la dimensione maggiore dell'ellisse, il che significa che questa può sempre essere contenuta nel cerchio che si otterrebbe tralasciando l'argomento *rapporto* (osservate la Figura 11.5).
Il seguente programma è una variazione del precedente che disegnava 25 cerchi. Questo programma disegna 25 ellissi alte e successivamente 25 ellissi larghe.

```
10 REM Ellissi
20 SCREEN 2
30 COLOR 2,6 : CLS
40 PRESET (68, 46)
50 FOR A = 1 TO 5
60 FOR B = 1 TO 5
70 CIRCLE STEP (20, 0),10,,,, 2
80 NEXT B
90 PRESET STEP (-100, 20)
100 NEXT A
110 TIME = 0
120 IF TIME < 500 THEN GOTO 120
130 CLS
140 PRESET (68, 46)
150 FOR A = 1 TO 5
160 FOR B = 1 TO 5
170 CIRCLE STEP (20, 0), 10,,,, .5
180 NEXT B
190 PRESET STEP (-100, 20)
200 NEXT A
210 TIME = 0
220 IF TIME < 500 THEN GOTO 220
```

Notate le tre virgole tra il raggio e il rapporto di allungamento; prendono il posto del colore e dei due angoli.

COLORARE LE FIGURE

Una delle prestazioni grafiche più avanzate dell'MSX-BASIC è la sua capacità di riempire un'area scelta con un colore. Finora abbiamo creato figure costituite da semplici linee; l'istruzione PAINT vi dà un semplice metodo per riempire un'area delimitata con un colore.
Nel modo 2, l'istruzione PAINT dipinge l'area dello stesso colore delle linee di contorno. La sintassi dell'istruzione PAINT è:

PAINT [STEP] (*x*, *y*) [,*colore*]

Gli argomenti STEP e *colore* non sono indispensabili. Il punto (*x*, *y*) è quello da dove iniziare a colorare. *colore* è il colore della linea di contorno; ne avete bisogno solamente se non è uguale al colore corrente di primo piano. Il programma che segue disegna un cerchio in (100, 100) con raggio 60 e lo riempie di colore.

```
10 REM Colorazione di un'area
20 SCREEN 2
30 COLOR 15 ,6 : CLS
40 CIRCLE (100, 100), 60
50 PAINT STEP (0, 0)
60 TIME = 0
70 IF TIME < 250 THEN GOTO 70
```

L'istruzione PAINT alla linea 50 sfrutta il fatto che l'istruzione CIRCLE lascia il cursore grafico nel centro del cerchio. Potete far iniziare l'azione di PAINT ovunque all'interno della vostra area. Per esempio, potete modificare la linea 50 in

```
50 PAINT (120, 80)
```

e il risultato sarebbe lo stesso.
L'unico aspetto rischioso nell'uso dell'istruzione PAINT è che, se la zona che dovete colorare non è completamente chiusa, se ci sono discontinuità anche di un solo pixel, PAINT riempie tutto lo schermo. Per verificarlo, modificate il precedente programma in modo da aggiungere un'apertura nel cerchio e guardate cosa succede:

```
10 REM Colorazione di un'area non circoscritta
20 SCREEN 2
30 COLOR 15 ,6 : CLS
40 CIRCLE (100, 100), 60
45 PRESET (100, 160)
50 PAINT (100, 100)
60 TIME = 0
70 IF TIME < 250 THEN GOTO 70
```

L'MSX-BASIC trova sempre l'apertura, non importa quanto sia piccola. L'apertura è costituita da pixel dello stesso colore dello sfondo, mentre la diagonale di unione tra i pixel non è considerata una discontinuità (vedi Figura 11.6).
Se volete disegnare dei rettangoli e riempirli di colore, non avete bisogno dell'istruzione PAINT. Esiste un altro argomento di LINE che riempie i rettangoli molto più velocemente: proprio come l'argomento B crea dei quadrati, l'argomento BF crea dei quadrati pieni. Quindi, l'istruzione:

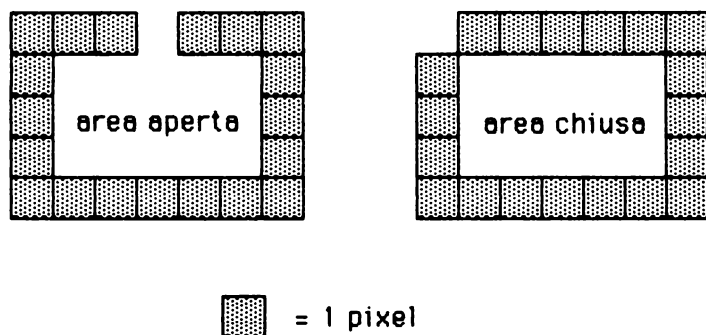



Figura 11.6 Aree da colorare

```
290 LINE - STEP (20, 50), 13, BF
```

disegna un rettangolo 20×50 riempito di color lavanda, a partire dalla posizione corrente del cursore grafico.

L'ISTRUZIONE DRAW

DRAW è un potente strumento grafico col quale è più facile disegnare figure che con una sequenza di molte istruzioni PSET e LINE. DRAW ha un proprio linguaggio con propri comandi, chiamato *Graphic Macro Language* o GML. I comandi GML si inseriscono in una stringa invece che su linee separate.
La sintassi di DRAW è molto semplice:

DRAW "*stringa*"

Ci sono molti comandi GML che possono essere inseriti nella stringa: li trovate elencati nella Figura 11.7. DRAW può ovviamente essere usato insieme ad altre istruzioni grafiche. Ogni comando può essere separato dagli altri nella stringa tramite un punto e virgola (;) ma nella maggior parte dei casi non è necessario; usate comunque un punto e virgola per sicurezza. I comandi si possono dare sia con lettere minuscole che maiuscole e potete mettere, se volete, degli spazi tra i comandi. Dopo ogni comando il cursore grafico viene riposizionato.
I comandi più semplici sono U, D, L ed R. Per disegnare un quadrato 35×20, dovete battere:

Comando	**Significato**
U*n*	Disegna *n* pixel in su
D*n*	Disegna *n* pixel in giù
L*n*	Disegna *n* pixel a sinistra
R*n*	Disegna *n* pixel a destra
E*n*	Disegna *n* pixel in su e a destra
F*n*	Disegna *n* pixel in giù e a destra
G*n*	Disegna *n* pixel in giù e a sinistra
H*n*	Disegna *n* pixel in su e a sinistra
B*comando*	Esegue *comando* senza disegnare
N*comando*	Ritorna dopo l'esecuzione di *comando*
M*x*,*y*	Si posiziona in (*x*,*y*)
A*n*	Ruota di (90**n*) gradi
S*n*	Riproduce in scala *n*/4
C*n*	Assegna il colore *n*
X*stringa*	Esegue i comandi contenuti in *stringa*

Figura 11.7 I comandi GML

```
DRAW "r35 d20 l35 u20"
```

I comandi E, F, G e H disegnano linee a 45 gradi. La Figura 11.8 mostra gli angoli usati da questi comandi. Per esempio, la seguente istruzione disegna una girandola:

```
DRAW "e5d5l5f5l5u5g5u5r5h5r5d5"
```

Il comando M è molto simile a PSET. Per ottenere uno spostamento assoluto si indicano le due coordinate dopo la M:

```
DRAW "m100,35;u10"
```

Spostamenti relativi si indicano mettendo un + o un − davanti alla coordinata *x*. Per esempio, per muovere il cursore a sinistra di 3 e giù di 10, date l'istruzione

```
DRAW "m-3,10"
```

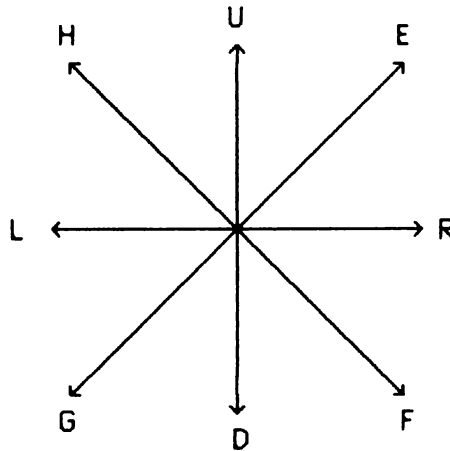



Figura 11.8 Comandi di spostamento

Per muovere a destra di 50 e su di 20, date l'istruzione

```
DRAW "m+50,-20"
```

Poiché l'argomento dell'istruzione DRAW è una stringa, è semplice porre la stringa in una variabile e usare questa con DRAW. Siccome le variabili stringa possono essere concatenate fra loro, potete creare facilmente dei disegni. Per esempio:

```
20 PW$ "e5d515f515u5g5u5r5h5r5d5"
...
250 DRAW PW$ + "m-20,-20" + PW$
```

I comandi B ed N sono in realtà dei prefissi per altri comandi GML. Il prefisso B dice a DRAW di non disegnare mentre viene eseguito il comando successivo. Per esempio, modificando la linea 250 del precedente esempio così:

```
250 DRAW PW$ + "bm-20,-20" + PW$
```

si elimina la linea tra le due girandole.
Il prefisso N dice a DRAW di ritornare al punto di inizio dopo il comando successivo. Per esempio, un semplice modo per disegnare una freccia che punta al cursore grafico, è con l'istruzione

```
DRAW "nl20nh20ng20"
```

Il cursore ritorna nel punto in cui si trovava prima dell'esecuzione dell'istruzione.
Il comando di colore, C, vi permette di selezionare il colore di primo piano. È come usare l'istruzione COLOR solamente con il primo argomento. Per esempio, per selezionare il nero come colore di primo piano e disegnare un quadrato, potete dare l'istruzione

```
280 DRAW "c1u10r10d10l10"
```

Il linguaggio GML ha due comandi estremamente interessanti e versatili: A e S. Il comando A cambia l'angolazione dell'intero disegno. "A0" è l'orientamento normale, "A1" ruota tutti i disegni che seguono di 90 gradi in senso orario, "A2" li ruota di 180 e "A3" di 270. Per vedere quanto sia utile il comando A, immaginate di avere impiegato un sacco di tempo per disegnare in GML una automobile, il cui piano di appoggio è parallelo alla base dello schermo. Più avanti nel programma, decidete di disegnarla lungo un lato dello schermo. Anziché dover riscrivere l'intera stringa potete semplicemente aggiungere all'inizio il comando "A1":

```
100 A$ = "...stringa..."
.
.
500 REM Disegna l'auto in posizione normale
510 DRAW A$
.
.
800 REM La ruota
810 DRAW "a1" + A$
```

Il comando S vi permette pure una grande versatilità, dandovi la possibilità di cambiare la scala del disegno. Dopo un comando S, la scala del disegno viene portata a un quarto del valore dell'argomento del comando S. Quindi, "S8" indica a DRAW di ingrandire di due volte le dimensioni, "S2" di ridurle a metà.
Continuando l'esempio dell'automobile, immaginate di avere fatto sì che A$ disegni un'automobile all'incirca delle dimensioni dello schermo. Più avanti nel programma avete bisogno di disegnare l'automobile ridotta a un quarto delle dimensioni e nell'angolo in basso a destra dello schermo. Ponendo che tutti i comandi M che si trovano nella stringa A$ usino spostamenti relativi, potete dare semplicemente l'istruzione:

```
1270 DRAW "m128,86;s1;" + A$
```

Il comando X non è molto utile, poiché vi permette solamente di usare

un'altra stringa nell'instruzione DRAW. È facile come concatenare le stringhe, come mostrato negli esempi precedenti. In ciascuno dei comandi GML che assumono un valore, al posto di questo potete usare un nome di variabile preceduto da un segno di uguale e da un punto e virgola. Per esempio se volete scrivere una stringa generica in GML per disegnare un quadrilatero potete usare:

```
"r=XO;d=YO;l=XO;u=XO"
```

Il modo migliore per sfruttare la versatilità dell'istruzione DRAW è di porre tutti i suoi comandi all'interno di variabili ed usare solo spostamenti relativi. Scoprirete che DRAW sveltirà l'esecuzione dei programmi rispetto a PSET e LINE.

11.3 Visualizzazione del testo in modo 2

È probabile vogliate inserire del testo nella visualizzazione di un programma grafico. Sfortunatamente, l'MSX-BASIC non ha un modo semplice per inserire un testo sullo schermo grafico. Il metodo qui illustrato si basa su comandi che non hanno nulla a che vedere con la grafica.
Prima di vedere come fare, diamo un'occhiata alle periferiche dell'MSX. Tenete presenti i seguenti punti:

- Una periferica si inizializza con l'istruzione OPEN. Questo comando dice all'MSX-BASIC che volete accedere alla periferica. Quando avete finito di usarla, potete dare l'istruzione CLOSE per dire al computer che non volete più accedervi.
- Alcune periferiche possono ricevere informazioni, altre ne possono trasmettere ed altre ancora possono fare entrambe le cose. Se una periferica può ricevere informazioni, si apre per l'output (ciò significa che il computer si prepara a mandare dati alla periferica); se può solo inviare informazioni, si apre per l'input; se può fare entrambe le cose si apre con accesso casuale.
- Molti comandi MSX-BASIC possono mandare o ricevere un'informazione a e da una periferica. Il modo piu diretto per mandare un'istruzione a una periferica è con l'istruzione PRINT e quello per riceverla è INPUT e LINE INPUT.
- Ogni periferica ha un nome. Per esempio, lo schermo grafico si chiama GRP: e la porta per cassette si chiama CAS:.
- Quando inizializzate una periferica, le assegnate un numero. Questo numero si utilizza in tutti i comandi MSX-BASIC con cui si comunica con la periferica.

Tutto ciò può sembrare molto complicato, ma vi sarà più chiaro dopo aver visto alcuni esempi. La periferica di cui ci siamo occupati fino adesso è lo schermo grafico. In breve, vogliamo riuscire a mandare del testo su di esso. L'MSX-BASIC sa che ogni testo mandato alla periferica GRP: deve essere stampato sullo schermo del modo 2.
Aprite GRP: per l'output visto che dovete solamente mandargli informazioni; non potete leggerne nessuna da esso. La sintassi di OPEN è:

OPEN "*nomefile*" FOR *modo* AS[#] *n*

In questo caso, *nomefile* è GRP: e *modo* è OUTPUT. *n* è il numero che assegnerete al file e che userete nei vostri comandi. Ogni file o periferica deve avere un numero diverso; 0 è un numero riservato per l'MSX-BASIC. Poiché non avete ancora usato l'istruzione OPEN, usate 1 per *n*. Quindi l'istruzione OPEN in questo caso è:

```
OPEN "GRP:" FOR OUTPUT AS #1
```

Per inviare informazioni allo schermo grafico abbiamo bisogno di un'istruzione PRINT: l'unica differenza tra questo PRINT e quelli che avete visto nel Capitolo 10 è che in questo caso bisogna aggiungere il numero della periferica seguito da una virgola subito dopo la parola PRINT. Per esempio, se avete usato la seguente linea nel modo 0 o per stampare del testo:

```
260 PRINT "Punteggio:"
```

dovete semplicemente aggiungere #1, dopo la parola PRINT:

```
260 PRINT #1,"Punteggio:"
```

Potete ora visualizzare informazioni sullo schermo grafico. Il cursore grafico si trova nell'angolo in alto a sinistra del primo carattere da stampare. Per esempio, per aggiungere una riga di testo in fondo all'esempio dei 25 cerchi, basta aggiungere l'istruzione OPEN:

```
15 OPEN "GRP:" FOR OUTPUT AS #1
```

e le linee

```
103 PRESET (90,150)
106 PRINT #1, "25 cerchi"
```

Se volete potete cambiare il colore del testo con

```
104 COLOR 15
```

Scrivere sullo schermo in questo modo presenta qualche svantaggio; quello che si deve ricordare è che ogni immagine viene solamente ricoperta dal testo e non cancellata. Se stampate su una zona dove esiste già un testo, il precedente non verrà cancellato. Inoltre, stampare gli spazi non cancella i disegni sottostanti.

11.4 Gli sprite

Se volete scrivere un gioco di animazione, gli sprite vi renderanno le cose molto più semplici. Gli sprite sono delle immagini speciali che potete controllare con il computer: si possono spostare per tutto lo schermo senza interferire con le altre immagini. Per esempio, potete muovere uno sprite sopra un disegno, ed esso non verrà cancellato dagli spostamenti dello sprite.
Per sfruttare al meglio i vantaggi degli sprite, è richiesta una conoscenza del funzionamento interno del computer, il che è ben oltre gli scopi di questo libro; ci limitiamo quindi a presentarvi i comandi relativi agli sprite e a fornirvi degli esempi circa il loro uso, per stimolarvi a ulteriori esperimenti. L'istruzione SCREEN, che abbiamo già visto, ha un secondo argomento che controlla gli sprite. La sua sintassi è

SCREEN *modo, dimsprite*

La variabile *dimsprite* deve essere 0, 1, 2 o 3; la Figura 11.9 dà il significato di questi valori. Qui considereremo sempre che il valore sia quello di default, 0. Ogni volta che date l'istruzione SCREEN, tutte le informazioni relative agli sprite vengono distrutte, quindi accertatevi di creare i vostri sprite dopo l'istruzione SCREEN. Ogni sprite è contrassegnato da un numero che potete assegnare voi: ci possono essere fino a 64 sprite, numerati da 0 a 63. Quando ci si riferisce ad uno sprite, lo si fa sempre per mezzo del suo numero.
L'immagine assegnata a uno sprite viene conservata in un'area di memoria riservata agli sprite. Per definire la forma dello sprite, non valgono i comandi grafici che conosciamo, dovete usare delle stringhe di dati e la funzione SPRITE$. Ogni sprite è lungo otto caratteri; per decidere come disegnarlo, il computer esamina i bit di ogni carattere (vedi Figura 11.10). Quando definite la forma di uno sprite, ogni bit con il valore 1 è un pixel acceso; ogni bit con il valore 0 è "trasparente": se collocate lo sprite sopra un disegno, potete vedere lo sfondo attraverso lo sprite nei punti dove c'è un bit che vale 0. Per disegnare lo sprite in un quadrato di 8×8 pixel, conviene tracciare uno schizzo su una griglia e poi usare l'operatore

Valore	Significato
0	Ogni sprite è 8x8 pixel e occupa 8 byte
1	Ogni sprite è 8x8 pixel e occupa 8 byte, ma sullo schermo è ingrandito di due volte
2	Ogni sprite è 16x16 pixel e occupa 32 byte
3	Ogni sprite è 16x16 pixel e occupa 32 byte, ma sullo schermo è ingrandito di due volte

Figura 11.9 Valori dell'argomento *dimsprite*

binario &B per assegnare ad ogni carattere una stringa di 8 caratteri. Per esempio la Figura 11.11 mostra uno sprite costituito da due quadrati concentrici. Il seguente programma inizializza la variabile S1$ al carattere che rappresenta lo sprite

```
100 B1$ = CHR$(&B11111111)
110 B2$ = CHR$(&B10000001)
120 B3$ = CHR$(&B10111101)
130 B4$ = CHR$(&B10100101)
140 S1$ = B1$ + B2$ + B3$ + B4$ + B4$ + B3$ + B2$
+ B1$
```

Figura 11.10 Definizione di uno sprite

Carattere 1	11111111
Carattere 2	10000001
Carattere 3	10111101
Carattere 4	10100101
Carattere 5	10100101
Carattere 6	10111101
Carattere 7	10000001
Carattere 8	11111111

Figura 11.11 Un esempio di sprite

Definita S1$, potete ora usare la funzione SPRITE$ per memorizzare il valore dello sprite. La sintassi della funzione SPRITE$ è:

SPRITE$ (*n*)=*stringa*

Poniamo di voler immagazzinare la stringa come immagine numero 5; diamo allora la seguente istruzione:

```
150 SPRITE$(5) = S1$
```

L'istruzione PUT SPRITE vi permette di posizionare lo sprite sullo schermo e di muoverlo. Potete anche cambiarne il colore e persino il numero dell'immagine che lo sprite rappresenta. La prima funzione di PUT SPRITE è assegnare il numero dello sprite e dell'immagine. La sua sintassi per l'inizializzazione è

PUT SPRITE *numerosprite ,,, immagine*

Per il nostro programma di esempio, assegnate il numero di sprite 1. Quindi l'istruzione è

```
160 PUT SPRITE 1,,,5
```

Quando volete muovere lo sprite sullo schermo, usate la seguente sintassi:

PUT SPRITE *numerosprite*, [STEP] (*x*, *y*) [,*colore*]

Gli argomenti STEP e *colore* non sono indispensabili: se date l'argomento *colore*, questo colorerà tutti i pixel che valgono 1. Per esempio per colo-

rare di grigio uno sprite e piazzarlo vicino al centro dello schermo, date l'istruzione

```
170 PUT SPRITE 1, (128, 96), 14
```

Siete ora pronti a muovere lo sprite sullo schermo con PUT SPRITE. Nel seguente programma potrete vedere tutto a colori. La prima parte del programma disegna uno sfondo complicato, ma la seconda (le linee dalla 100 alla 170) è costituita dalle linee dei precedenti esempi e la terza parte vi consente di muovere lo sprite con i tasti di controllo del cursore. Per muovere lo sprite, premete uno dei tasti con la freccia; per fermare il programma, premete il tasto RETURN.

```
10 REM Esempio di sprite
20 SCREEN 2, 0
30 COLOR 11, 8 : CLS
40 REM Predispone lo sfondo
50 GOSUB 500
60 GOSUB 600
100 B1$ = CHR$(&B11111111)
110 B2$ = CHR$(&B10000001)
120 B3$ = CHR$(&B10111101)
130 B4$ = CHR$(&B10100101)
140 S1$ = B1$ + B2$ + B3$ + B4$ + B4$ + B3$ + B2$
+ B1$
150 SPRITE$(5) = S1$
160 PUT SPRITE 1, , , 5
170 PUT SPRITE 1, (128, 96), 14
200 REM Routine di input da tastiera
210 K$ = INPUT$(1)
220 REM Controllo del RETURN
230 IF K$ = CHR$(13) THEN GOTO 390
240 ON (ASC(K$)-27) GOSUB 270, 300, 330, 360
250 PUT SPRITE 1, STEP (X, Y)
260 GOTO 200
270 REM Movimento a destra
280 X = 2 : Y = 0
290 RETURN
300 REM Movimento a sinistra
310 X = -2 : Y = 0
320 RETURN
330 REM Movimento verso l'alto
340 X = 0 : Y = -2
350 RETURN
360 REM Movimento verso il basso
370 X = 0 : Y = 2
380 RETURN
390 REM Uscita dal programma
400 COLOR 15, 4
```

```
410 END
500 REM Rettangoli colorati
510 A = RND(-TIME)
520 FOR I = 1 TO 45
530 LINE (RND(1)*255, RND(1)*195) - STEP (20, 40),
(I MOD 15), BF
540 NEXT I
550 RETURN
600 REM Inserimento del testo
610 REM sulla prima riga
620 LINE (0,0) - (255,8), 1, BF
630 OPEN "GRP:" FOR OUTPUT AS #1
640 PSET (15, 0), 3
650 PRINT #1, " Usare le frecce per muovere"
660 RETURN
```

Se ve la sentite di sperimentare altre forme di sprite, provate a modificare le linee dalla 100 alla 150. Per esempio le linee che seguono creano un quadrato a scacchiera.

```
100 B1$ = CHR$(&B10101010)
110 B2$ = CHR$(&B01010101)
140 S1$ = B1$ + B2$ + B1$ + B2$ + B1$ + B2$ + B1$
+ B2$
```

Se vi interessano i marziani, quello che segue può essere utilizzato in un gioco:

```
100 B1$ = CHR$(&B00011000) : B5$ = B1$
110 B2$ = CHR$(&B00100100) : B4$ = B2$
120 B3$ = CHR$(&B01011010)
125 B6$ = CHR$(&B00100100)
130 B7$ = CHR$(&B11000011)
135 B8$ = CHR$(&B01000010)
140 S1$ = B1$ + B2$ + B3$ + B4$ + B5$ + B6$ + B7$
+ B8$
```

Ci sono altre due istruzioni che possono essere usate dai programmatori esperti che usano gli sprite. L'istruzione SPRITE permette di attivare una speciale funzione dell'MSX-BASIC, che salta ad una subroutine se due sprite si toccano. La linea alla quale salta l'MSX-BASIC è determinata dall'istruzione ON SPRITE GOSUB.

Capitolo

Il suono 12

Con un computer MSX, con semplici comandi potete scrivere programmi che producono suoni di alta qualità.
Alcuni computer MSX hanno caratteristiche musicali particolarmente potenti; in tutti comunque troverete che la musica (e il rumore) che potete creare è di una qualità sorprendentemente alta.
Tre caratteristiche rendono il sistema musicale MSX così potente:

1. Buona parte degli home computer hanno una sola voce, possono cioè produrre un suono alla volta. I computer MSX hanno tre voci. Questo rende la musica molto più vivace, poiché è facile creare delle armonie.
2. L'MSX-BASIC può suonare la musica mentre gira un programma: potete così comporre una musica di sottofondo da suonare durante l'esecuzione del programma, senza che questa influisca sulla velocità di quest'ultimo.
3. È possibile inoltre variare le caratteristiche principali dei suoni prodotti. Potete riprodurre gli strumenti musicali o creare rumori complessi per i giochi.

Prima di cominciare a provare i programmi di questo capitolo, dovete collegare al vostro sistema un altoparlante: il vostro computer non ne ha uno proprio. Potete usare un televisore o un monitor con altoparlante, oppure dovete avere uno spinotto di uscita audio per collegare il computer allo stereo. Se usate un televisore, assicuratevi che il volume sia abbastanza alto da poter sentire la musica che producete.

12.1 Un semplice beep

Prima di ascoltare tutti i meravigliosi suoni disponibili in MSX-BASIC, vediamo come si produce il suono più semplice. L'istruzione BEEP fa esattamente quello che dice il nome: produce un corto beep nell'altoparlante. È utile quando volete inviare un avvertimento o sottolineare un messaggio.
La sintassi di BEEP è molto semplice:

BEEP

Per esempio:

```
400 PRINT "Non farlo!"
410 BEEP
```

Potete anche produrre un beep mediante il carattere il cui codice ASCII è 7, non preceduto dal carattere ASCII 1 (vedi Appendice D); per esempio

```
400 PRINT "Non farlo!"; CHR$(7)
```

12.2 La musica

Il modo più semplice per scrivere musica è usare l'istruzione PLAY. Questa è simile a DRAW: ha delle stringhe come argomenti, che sono scritte in un particolare linguaggio chiamato *Music Macro Language* (MML). Se conoscete già le scale e le nozioni di base della musica troverete molto semplice l'utilizzo dei comandi MML. La sintassi dell'istruzione PLAY è molto semplice:

PLAY *"stringa1"*, *"stringa2"*, *"stringa3"*

Le stringhe sono i comandi in MML per ognuna delle tre voci. Se state usando delle voci che non siano la numero 1, dovete comunque mettere le virgole di separazione. Per esempio, per usare la voce 3 con una stringa chiamata BACH$, date il comando

```
PLAY , , BACH$
```

Le regole dell'MML sono molto simili a quelle del GML che avete imparato nel precedente capitolo. Potete separare i comandi con uno spazio o

con un punto e virgola, o potete usare variabili MSX-BASIC al posto degli argomenti numerici mettendo un segno di uguale prima del nome e un punto e virgola dopo. Potete dare i comandi sia in lettere maiuscole che minuscole. I comandi MML sono riassunti nella Figura 12.1

Comando	**Significato**
C*n*	Do (suonato per 1/*n* di battuta)
D*n*	Re "
E*n*	Mi "
F*n*	Fa "
G*n*	Sol "
A*n*	La "
B*n*	Si "
R*n*	Pausa (per 1/*n* di battuta)
#	Diesis
+	Diesis
-	Bemolle
.	Estende la durata del 50%
O*n*	Ottava (da 1 a 8, 4 per default)
L*n*	Durata di ogni nota da quel punto in poi (da 1 a 64, 4 per default)
T*n*	Ritmo (da 32 a 255, 120 per default)
V*n*	Volume (da 0 a 15, 8 per default)
N*n*	Valore di ogni singola nota (da 1 a 96)
M*n*	Periodo dell'inviluppo (da 1 a 65535, 255 per default)
S*n*	Forma d'onda (da 1 a 15, 1 per default)
X*stringa*	Esegue i comandi contenuti in *stringa*

Figura 12.1 Comandi MML

SEMPLICI NOTE

Suonare le sette note è abbastanza facile: dovete solamente indicarle con il loro nome in notazione anglosassone. Per esempio, per suonare le prime note del famoso brano "Fra Martino Campanaro", nella voce 1, date semplicemente l'istruzione:

```
PLAY "cdeccdecefgg"
```

I diesis e i bemolle sono facili quanto le note di base. Per alzare di un tono il precedente motivetto, date il comando

```
PLAY "c#d#fc#c#d#fc#ff#g#g#"
```

oppure

```
PLAY "c+d+fc+c+d+fc+ff+g+g+"
```

oppure

```
PLAY "d-e-fd-d-e-fd-fg-a-a-"
```

Per indicare la durata delle note, viene utilizzato il numero di note uguali che possono stare in una battuta.
Il valore di default della durata di una nota è 4, il che significa che essa viene suonata come una nota da un quarto in una battuta da 4/4. Un valore di 1 significa 4/4, 2 significa 2/4 e così via. Per suonare il brano originale in note da 1/3, potete scrivere

```
PLAY "c3d3e3c3c3d3e3c3e3f3g3g3"
```

Potete indicare durate da 1 (4/4) a 64 (1/64). Il comando R indica una pausa, e può essere usato come le altre note.
Avrete notato che l'MSX-BASIC non stacca i due Do della prima frase. Lo stacco fra una nota e l'altra può essere avvertito solo tra note diverse. Questa è una sfortunata limitazione, ma può essere raggirata lavorandoci sopra. Vedremo più avanti come inserire piccole pause nelle ripetizioni della stessa nota.

MUSICA PIÙ COMPLESSA

Ogni ottava va da C (Do) a B (Si). Le ottave vengono numerate da 1 a 8 e, mancando indicazioni specifiche, viene assunta per default l'ottava nume-

ro 4. La suddivisione in ottave può talvolta creare dei problemi; per esempio, se abbassate il vostro motivetto di due note, otterrete

```
PLAY "abc#aabc#ac#dee"
```

Quando viene eseguito, suona sbagliato perché A e B (La e Si) sono di un'ottava troppo alta rispetto alle altre note. Potete modificarlo facilmente alzando di un'ottava le note A e B:

```
PLAY "o3abo4c#o3aabo4c#o3ao4c#dee"
```

Usare due voci è facile come usarne una. Per esempio, le note iniziali di un altro famoso brano, possono essere scritte

```
PLAY "fedc", "ggbo5co4"
```

Il comando "o4" ripristina l'ottava di partenza. È facile dimenticarsi di fare cose come queste, ma il computer si ricorda dell'ultima ottava selezionata per una voce dopo l'esecuzione dell'istruzione PLAY. Per verificarlo, togliete "o4" dalla voce 2 e date il comando due volte. Noterete che la seconda volta, la seconda nota inizia un'ottava più alta. Il comando L predispone la velocità di esecuzione per le note per le quali non è specificata una lunghezza; il valore convenzionale è quattro. Potete usare questo comando se volete incrementare il ritmo. Per esempio, per suonare tutto con un ritmo più veloce del doppio, date l'istruzione

```
PLAY "l8"
```

Ricordate poi di ripristinare il valore iniziale dando un comando "l4".
Il ritmo può essere modificato anche con il comando T. Il ritmo standard è 120, che significa 120 note da un quarto al minuto. Potete predisporre il ritmo a un qualsiasi valore tra 32 (lento) e 255 (veloce). Potete inoltre modificare il volume con il comando V. Inizialmente è posto a 8, ma potete scegliere un valore qualsiasi tra 0 (silenzio) e 15 (massimo).
Se preferite indicare le note con i numeri anziché con le lettere, usate il comando N. N0 è la nota più bassa mentre N96 è la più alta; il Do dell'ottava centrale, che è O4C, diventa N36. Ciò in realtà può essere utile solo se usate delle variabili. Per esempio, per suonare le note (compresi i semitoni) in sedicesimi dell'ottava che incomincia con O4C, potete dare le istruzioni

```
420 FOR I = 36 TO 48
430 PLAY "n=i;16"
440 NEXT I
```

I comandi M ed S modificano l'onda sonora prodotta e vengono descritti nel prossimo paragrafo. Essi saranno utili solo se avete qualche nozione sulla generazione dei suoni. Potete anche notare che il comando S si sostituisce al comando V, per cui non potete regolare il volume dopo aver usato il comando S. Il comando X vi permette semplicemente di usare un'altra stringa all'interno del comando PLAY mediante l'inserimento di una variabile. Il suo uso è facile: la variabile deve essere preceduta da X e seguita da un punto e virgola.
Potete controllare se in un programma viene suonata della musica con la funzione PLAY. La sintassi è

PLAY (*numerovoce*)

e risulterà vera se sta suonando della musica sulla voce indicata da *numerovoce*; per esempio:

```
230 IF PLAY(1) THEN GOTO 230
240 PRINT "Non viene suonato nulla sulla voce 1"
```

Usate *numerovoce*=0 per verificare se qualcuna delle voci è attiva.

COME FUNZIONA L'ISTRUZIONE PLAY

Tutte le volte che date l'istruzione PLAY, l'MSX-BASIC mette in lista d'attesa le note in un buffer musicale e poi continua a svolgere il programma mentre suona la musica. Infatti anche se il vostro programma è finito e l'MSX-BASIC ha già dato il messaggio "Ok", il buffer musicale continua a suonare fino a svuotarsi. Se date una seconda istruzione PLAY, l'MSX-BASIC aggiunge le note al buffer. Supponete per esempio che U1$ e U2$ contengano dei lunghi brani di musica in MML. Se eseguite le seguenti istruzioni

```
140 PLAY U1$
150 PRINT "Tieni duro......."
160 PLAY U2$
170 PRINT "Continua a tener duro......"
```

la prima istruzione PRINT viene eseguita mentre viene suonata la prima o la seconda nota di U1$ e così la seconda PRINT. La musica di U2$ inizia dopo che U1$ è finita. Anche se U2$ inizia immediatamente, potete sentire una brevissima pausa nella musica dopo la fine di U1$. Potete usare queste interruzioni a vostro vantaggio: esse sopperiscono all'impossibilità di differenziare le note ripetute. Fate girare il seguente programma e notate la differenza tra le due linee:

```
10 REM Esempio di interruzione
20 REM Continuo
30 PLAY "cdeccdecefggr1"
40 REM Con l' interruzione
50 PLAY "cdec" : PLAY "cdecefgg"
```

Questo è l'espediente cui ci si riferiva precedentemente in questo capitolo; è chiaramente un sistema poco elegante, ma funziona.
Se interrompete un programma che sta suonando musica premendo i tasti CTRL STOP, inserendo uno STOP nel programma o commettendo un errore durante l'esecuzione, l'MSX-BASIC emette un beep e interrompe il suono. Poi ripristina tutti i parametri musicali: O, L, T, V, M e S vengono riportati al loro valore di default. Anche un'istruzione BEEP ferma la musica e ripristina i parametri.

12.3 Effetti sonori

L'istruzione SOUND dà ai programmatori esperti l'accesso a tutte le parti del generatore di suoni MSX. Questo paragrafo presuppone che cono-

Numero	Funzione
0	Tono della voce A, byte basso
1	Tono della voce A, byte alto
2	Tono della voce B, byte basso
3	Tono della voce B, byte alto
4	Tono della voce C, byte basso
5	Tono della voce C, byte alto
6	Controllo del generatore di rumore (0-31)
7	Abilitazione voce e rumore
8	Controllo ampiezza voce A
9	Controllo ampiezza voce B
10	Controllo ampiezza voce C
11	Periodo dell'inviluppo, byte basso
12	Periodo dell'inviluppo, byte alto
13	Forma dell'inviluppo

Figura 12.2 Registri dell'istruzione SOUND

sciate come viene generato un suono e sappiate lavorare con i registri dei chip. Con SOUND potete creare rumori ed effetti sonori fantasiosi.
L'istruzione SOUND vi permette di predisporre i valori dei registri del chip sonoro all'interno del vostro computer MSX. La sua sintassi è:

SOUND *registro, valore*

registro è un numero di registro da 0 a 16; *valore* è il numero che volete immagazzinare nel registro da un byte. La Figura 12.2 riassume i registri (notate che le voci sono indicate con le lettere A, B e C anziché 1, 2 e 3 come nell'istruzione PLAY).
I registri 0 e 1 formano un numero di 12 bit (i quattro bit superiori del registro 1 non vengono utilizzati) che rappresenta il tono della voce A; i registri 2 e 3 formano la coppia per la voce B e i registri 4 e 5 sono la coppia per la voce C. Date a questi il valore del tono desiderato. Per determinare il tono usate l'equazione

tono = 124797/frequenza

Quindi per creare un suono di 440 Hz il tono vale 284. Inserite i valori nei registri:

```
SOUND 0, 29
SOUND 1, 1
```

Il registro 6 controlla il periodo dell'emissione e genera i diversi effetti in base ai valori di tutti gli altri registri. Se state generando effetti sonori, specialmente quelli nei quali si impiegano trilli, provate a modificare questo registro.
Il registro 7 serve ad attivare le capacità di suono o di rumore delle voci. La Figura 12.3 mostra come è suddiviso il registro 7. Per accendere una voce o per aggiungere la componente di rumore in quella voce, ponete 0 nel bit corrispondente. Il valore 1 invece spegne quella voce. Per spegnere la produzione di rumore ed accendere le voci A e B, date l'istruzione:

```
SOUND 7, &B00111100
```

I registri dall'8 al 10 verificano se volete agire sul volume o sull'inviluppo e la forma d'onda. La Figura 12.4 mostra come usare ognuno di questi registri. I primi tre bit vengono ignorati. Se intendete agire sul volume, i bit da 1 a 4 devono assumere un valore da 0 (piano) a 15 (forte).
I registri 11 e 12 contengono l'inviluppo e il registro 13 la forma d'onda. Per determinare la frequenza dell'inviluppo usate l'equazione

frequenza = 7799.8/periodo

	rumore C	rumore B	rumore A	suono C	suono B	suono A
bit	6	5	4	3	2	1

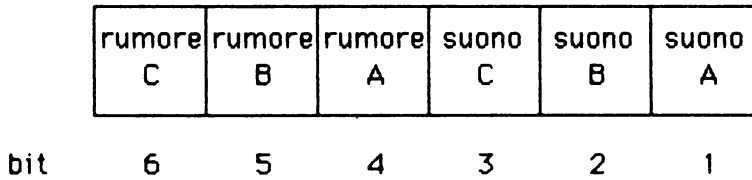



Figura 12.3 Suddivisione del registro 7

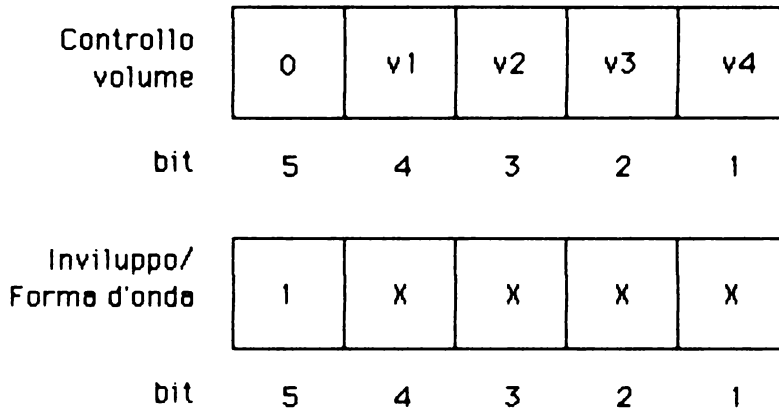



Figura 12.4 Suddivisione dei registri dall'8 al 10

La Figura 12.5 mostra le forme d'onda che potete usare. Notate che vengono usati solo i quattro bit inferiori del registro 13.
Con l'istruzione SOUND potete generare molti suoni strani ed interessanti. Provate questo programma:

```
10 REM Il rumore di una jeep
20 SOUND 6, 10
30 SOUND 7, &B00110110
40 SOUND 8, &B00000100
50 SOUND 1, 5
60 TIME = 0
```

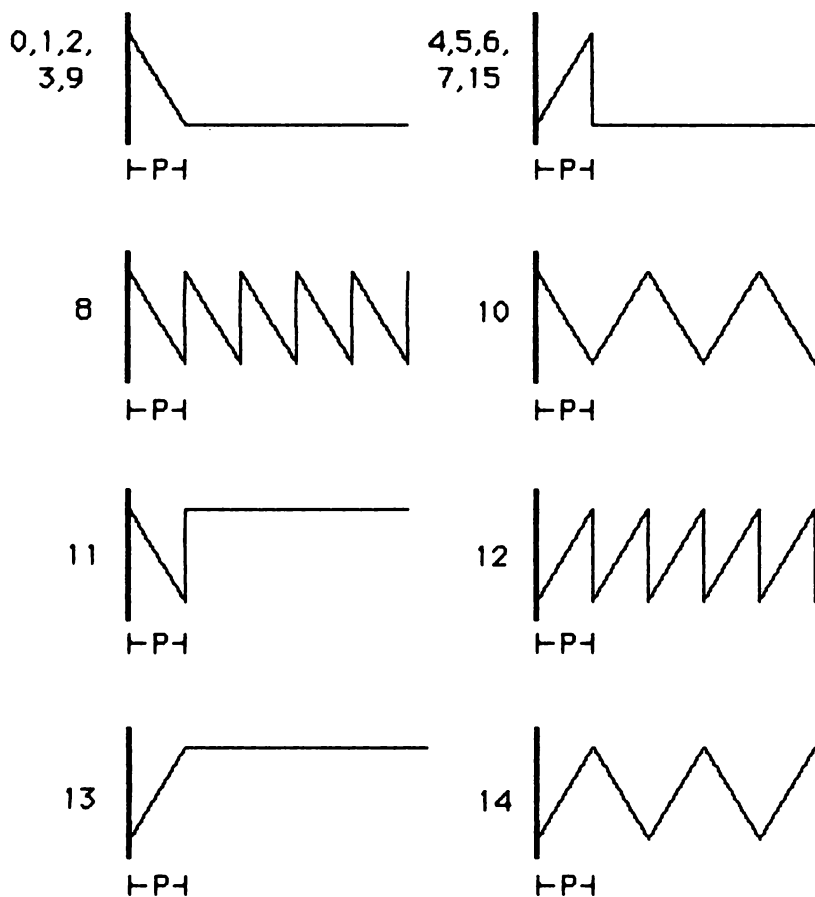



Figura 12.5 Forme d'onda del registro 13

```
70 IF TIME < 100 THEN GOTO 70
80 BEEP
```

Con l'istruzione SOUND si possono eseguire delle scale crescenti e decrescenti meglio che con l'istruzione PLAY. Per esempio

```
10 REM Toni puri
20 SOUND 1, 0
30 SOUND 7, &B00110110
40 SOUND 8, &B00000100
50 FOR X = 255 TO 0 STEP -1
60 SOUND 0, X
70 NEXT X
80 BEEP
```

Continuando gli esperimenti con l'istruzione SOUND scoprirete una vasta gamma di rumori e suoni interessanti: il risultato può essere molto gratificante.

Capitolo

Dispositivi di I/O 13

Negli ultimi tre capitoli abbiamo visto che il computer MSX può fare ben più che una semplice elaborazione di numeri: esso visualizza e recepisce informazioni, crea grafica, suona musica e produce rumori. Questo capitolo tratta dei procedimenti e delle periferiche di input e di output che incrementano ulteriormente le capacità del computer MSX.
Le più comuni periferiche degli home computer sono i joystick, indispensabili per giocare ai videogame, e i drive.

13.1 I joystick

Ogni computer MSX ha almeno una porta per un joystick del tipo compatibile Atari; buona parte ne hanno due. I joystick hanno otto posizioni (su, giù, a sinistra, a destra e le quattro posizioni in diagonale) e un interruttore o pulsante da premere. Due funzioni MSX-BASIC leggono le posizioni del joystick e del pulsante.
Se non avete un joystick l'MSX-BASIC può simularne le funzioni attraverso i tasti di controllo del cursore. Questi tasti non sono precisi e divertenti da usare come un joystick, ma sono ugualmente utili: le quattro direzioni principali sono indicate dalle frecce, lo spostamento in diagonale si ottiene premendo contemporaneamente due tasti adiacenti e la barra dello spazio sostituisce il pulsante.
La funzione STICK legge le posizioni del joystick e restituisce un valore numerico che varia in base alla direzione verso cui viene spinto il joystick. La sua sintassi è

Numero	Significato
0	Tasti di controllo del cursore
1	Joystick nella porta 1
2	Joystick nella porta 2

Figura 13.1 Porte del joystick

STICK (*n*)

dove *n* è il numero del joystick che state usando. La Figura 13.1 mostra i valori di *n* e la Figura 13.2 mostra i valori che può assumere la funzione STICK.
La funzione STRIG assume il valore −1 se il pulsante è premuto e 0 in caso contrario. La sua sintassi è

STRIG (*n*)

dove *n* è il numero del joystick che state usando, proprio come nella funzione STICK.

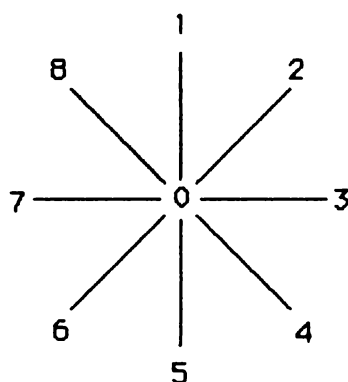



Figura 13.2 Valori della funzione STICK

Poiché entrambe le funzioni STICK e STRIG assumono il valore 0 se non sta accadendo nulla, potete utilizzarle direttamente in espressioni logiche senza dover fare ulteriori confronti. Per esempio la linea che segue vi dice se state premendo il pulsante del joystick 1:

```
IF STRIG(1) THEN PRINT "BANG! BANG!"
```

Per vedere come queste funzioni agiscono in un programma sostituiamo le seguenti linee alle linee dalla 200 alla 400 nel programma di spostamento dello sprite del Capitolo 11. La variabile JN può essere posta uguale a 0 per usare i tasti del cursore, oppure a 1 o 2 per usare i joystick. Premendo il pulsante si esce dal programma.

```
200 REM Input da joystick o da tasti cursore
205 JN = 2
210 IF STRIG(JN) THEN GOTO 310
220 D = STICK(JN)
230 IF (D = 0) THEN GOTO 210
240 IF (D = 1) THEN Y = -2
250 IF (D = 3) THEN X = 2
260 IF (D = 5) THEN Y = 2
270 IF (D = 7) THEN X = -2
290 PUT SPRITE 1, STEP (X, Y)
295 X = 0 : Y = 0
300 GOTO 200
310 REM Uscita dal programma
320 COLOR 15, 4
330 END
```

Per gestire l'uso dei joystick nella programmazione l'MSX-BASIC ha due interessanti istruzioni: ON STRIG GOSUB e STRIG che permettono all'MSX-BASIC di individuare l'attimo in cui viene premuto il pulsante, a prescindere da dove è arrivata l'esecuzione del programma. Se il programma utilizza questi comandi e l'utente preme il pulsante, l'MSX-BASIC interrompe qualsiasi cosa stia facendo e salta alla subroutine specificata (procedure di questo tipo vengono dette procedure di *interrupt*). Ciò significa che il programma può svolgere altre funzioni MSX-BASIC senza dover controllare costantemente la funzione STRIG.
Per ottenere una procedura di interrupt usate ON STRIG GOSUB. La sua sintassi è:

ON STRIG GOSUB *subr0, subr1, subr2*

Questa istruzione dice all'MSX-BASIC quale subroutine eseguire se viene premuto il pulsante 0, 1 o 2. Ogni argomento rappresenta il numero della linea d'inizio della relativa subroutine.

Per attivare la gestione dell'interrupt dovete però anche usare l'istruzione STRIG; ON STRIG GOSUB indica infatti solamente dove andare e non di iniziare a cercare gli interrupt. Le possibili sintassi di STRIG sono le seguenti:

STRIG (*n*) ON
STRIG (*n*) OFF
STRIG (*n*) STOP

L'argomento ON dice all'MSX-BASIC di cercare gli interrupt. Per far cessare la ricerca usate l'argomento OFF. L'argomento STOP dice all'MSX-BASIC di continuare la ricerca degli interrupt ma di non fare nulla nel caso ne incontrasse uno. Se più avanti date un'istruzione STRIG ON ed è sopravvenuto un interrupt, l'MSX-BASIC salta immediatamente alla subroutine.
L'esempio che segue predispone la subroutine per la barra dello spazio e per il pulsante 1:

```
520 ON STRIG GOSUB 800, 900
530 STRIG(0) ON
540 STRIG(1) ON
............
800 REM Barra di spazio
............
860 RETURN
900 REM Pulsante #1
............
960 RETURN
```

13.2 File su disco

Se avete un drive, potete usarlo anche per funzioni che non siano la semplice memorizzazione di programmi MSX-BASIC: potete scrivere e leggere file contenenti dati. Se non avete un drive, leggete comunque questo paragrafo: i vantaggi offerti dall'uso dei file sono tali che potreste convincervi a comprare un drive dopo averne compreso meglio la funzione.

APRIRE E CHIUDERE I FILE

L'MSX-BASIC in pratica, tratta un file come una periferica, per cui quanto abbiamo detto nel Capitolo 11 su queste ultime si può applicare ai file. Quindi, per iniziare a lavorare con un file dovete dare un'istruzione

OPEN. Il nome della periferica nell'istruzione OPEN viene sostituito dal nome del file (per i nomi dei file, leggete il Capitolo 18). Un esempio di istruzione che apre un file è:

```
30 OPEN "PROG1.DAT" FOR INPUT AS #2
```

L'MSX-BASIC vi consente di avere fino a 15 periferiche aperte nello stesso momento: dovete usare la pseudovariabile MAXFILES=, la cui sintassi è:

MAXFILES=*n*

per dire all'MSX-BASIC di riservare dello spazio nella memoria per gestire *n* periferiche; il valore di default è 1. Notate che l'uso di MAXFILES= comporta l'azzeramento di tutte le variabili in memoria, perciò usatela solo nelle primissime linee del programma.
Quando usate l'istruzione OPEN con i file, avete quattro possibilità per l'argomento FOR elencate nella Figura 13.3. Queste scelte sono molto importanti dato che una scelta sbagliata può provocare la cancellazione del file. Notate che se aprite un file con FOR OUTPUT e quel file è già esistente sul disco, esso viene cancellato.
L'istruzione CLOSE chiude un file e ne libera il numero di periferica. La sua sintassi è

CLOSE [*numfile1*], [*numfile2*]

Non siete obbligati a specificare i numeri delle periferiche: se date l'istruzione CLOSE senza argomenti, vengono chiuse tutte le periferiche. Dovete essere sicuri alla fine del programma di aver chiuso tutti i file,

Argomento	**Significato**
INPUT	Legge dal file
OUTPUT	Scrive sul file cancellandone il precedente contenuto
APPEND	Aggiunge dati alla fine del file
(non specificato)	Accesso casuale: legge e scrive record di lunghezza stabilita

Figura 13.3 Valori dell'argomento FOR

perché in alcuni casi l'MSX-BASIC non scrive l'ultima informazione nel file finché non è chiuso. Anche l'istruzione END chiude tutti i file ancora aperti, quindi concludere un programma con END è come dare l'istruzione CLOSE.

LETTURA DI UN FILE

Se avete intenzione di leggere informazioni da un file, questo deve essere aperto con un argomento FOR INPUT. Per leggere un file potete usare le istruzioni INPUT o LINE INPUT seguite da "*#n*", come

```
INPUT #2, GL$
```

Potete usare anche la funzione INPUT$; in questo caso il numero del file segue la lunghezza della stringa. Per esempio,

```
GL$ = INPUT$(5, 2)
```

legge cinque caratteri dalla periferica 2.
Questi comandi funzionano nello stesso modo di quelli esaminati nel Capitolo 10, ma non visualizzano nulla sullo schermo. Supponete per esempio che la prima linea del file di dati PRIMO.DAT sia

```
12, 17, Mario Bianchi
```

e che la prima linea del file di dati SCND.DAT sia

```
19, 2, Piero Rossi
```

Scrivete il seguente programma:

```
10 REM Esempio di lettura file
20 MAXFILES= 2
30 OPEN "PRIMO.DAT" FOR INPUT AS #1
40 OPEN "SCND.DAT" FOR INPUT AS #2
50 PRINT "La prima linea del primo file e':"
60 LINE INPUT #1, A$
70 PRINT A$
80 PRINT "Il primo numero nel secondo file e':";
90 INPUT #2, B
100 PRINT B
110 CLOSE
```

Il programma stampa

```
La prima linea del primo file e':
12, 17, Mario Bianchi
Il primo numero nel secondo file e': 19
```

Se usate un'istruzione INPUT che cerca di leggere oltre la fine di un file, l'MSX-BASIC vi dà il messaggio di errore "Input past end". Per evitare tutto questo, usate la funzione EOF (*End Of File*, cioè fine del file). Questa funzione vale −1 (vero) se vi trovate alla fine del file, oppure 0 in caso contrario. La sua sintassi è:

EOF (*n*)

Qui *n* è il numero della periferica che volete controllare. Il seguente programma legge un file di testo e lo stampa sullo schermo.

```
10 REM Stampa un file di testo
20 OPEN "ES.TXT" FOR INPUT AS #1
30 IF EOF(1) THEN GOTO 70
40 LINE INPUT #1, V$
50 PRINT V$
60 GOTO 30
70 REM Fine delle linee
80 END
```

SCRITTURA DI UN FILE

Se avete intenzione di scrivere delle informazioni su un file, dovete aprirlo con FOR OUTPUT o FOR APPEND. Come sottolineato in precedenza, usando FOR OUTPUT si cancella qualsiasi file con lo stesso nome prima che questo venga aperto. L'argomento FOR APPEND apre un file per l'output, ma si limita ad aggiungere dati alla fine del file senza toccare le altre informazioni già presenti in esso. Usate l'istruzione PRINT come avete fatto nel Capitolo 11:

```
10 REM Scrittura file
20 MAXFILES= 2
30 OPEN "OUT1.FIL" FOR OUTPUT AS #1
40 OPEN "OUT2.FIL" FOR OUTPUT AS #2
50 PRINT #1, "Parte 1"
60 PRINT #2, "Parte 1"
70 CLOSE
80 REM Attenzione alle seguenti due linee
90 OPEN "OUT1.FIL" FOR OUTPUT AS #1
100 OPEN "OUT2.FIL" FOR APPEND AS #2
110 PRINT #1, "Parte 2"
120 PRINT #2, "Parte 2"
130 CLOSE
```

Dopo l'esecuzione del programma, OUT1.FIL contiene solamente

```
Parte 2
```

perché avete usato FOR OUTPUT alla linea 90, ma OUT2.FIL contiene

```
Parte 1
Parte 2
```

poiché lo avete aperto con FOR APPEND alla linea 100.
Potete usare la funzione LOF per scoprire quanti caratteri sono stati scritti o letti da un file. La sua sintassi è:

LOF (*numfile*)

È utile per sapere quanto spazio è rimasto nel file.
Prima di aprire un file per l'output, potete controllare che il disco abbia spazio sufficiente per contenerlo, per mezzo della funzione DSKF che indica il numero di blocchi ancora disponibili (sul manuale del vostro drive è indicato il numero di byte per blocco).
La sintassi della funzione DSKF è:

DSKF (*numdrive*)

Il valore di *numdrive* è 0 per il drive corrente, 1 per "A:" o 2 per "B:".
Per esempio, potete includere le seguenti linee in un programma.

```
.........
480 IF DSKF(O) < 3 THEN GOTO 2000
490 OPEN "DATI.OUT" FOR OUTPUT AS #3
.........
2000 REM Il disco e' quasi pieno
.........
```

FILE AD ACCESSO CASUALE

Un file ad accesso casuale può essere sia letto che scritto; potrebbe sembrare più pratico degli altri file di solo input o di solo output, ma presenta molti limiti che ne rendono complicato l'uso. L'ostacolo principale è che bisogna usare dei record di lunghezza fissa (i record sono stati descritti nel Capitolo 2).
In breve, per usare un file ad accesso casuale dovete decidere le dimensioni di ogni record prima di usare il file. Ogni record è costituito da campi come quelli mostrati in Figura 13.4. Non solo i record ma anche i

	campo 1	campo 2	campo 3
record 1			
record 2			
record 3			
record 4			

Figura 13.4 Record e campi in un file ad accesso casuale

campi devono avere dimensioni prestabilite. Questo provoca uno spreco di spazio sul disco se i vostri record contengono stringhe, poiché dovete assegnare al campo una lunghezza pari alla più lunga stringa possibile. Per esempio vi potrebbe capitare di dover assegnare il nome di una città a un campo lungo 20 caratteri anche se la maggior parte delle città ha dei nomi più brevi di 10 caratteri.
Quando accedete ai dati del vostro file, l'MSX-BASIC legge un intero record alla volta e immagazzina le informazioni in un buffer, che è una serie di variabili stringa che avete specificato. Per aggiungere o modificare dati, usate delle funzioni specifiche per immagazzinare i dati nel buffer; da qui viene modificato nel file un intero record alla volta.
Per aprire un file ad accesso casuale date l'istruzione OPEN senza l'argomento FOR, ma con l'argomento LEN= alla fine. La sintassi è

OPEN "*nomefile*" AS [#]*n* LEN=*lungrec*

L'argomento *lungrec* è la lunghezza, in caratteri, di ogni record e deve essere un numero tra 1 e 256. Ogni campo viene immagazzinato come una stringa, perciò la somma delle lunghezze dei campi è la lunghezza del record. Se volete memorizzare dati numerici potete convertirli in stringhe per risparmiare spazio nel file. Ricordate che:

- Gli interi occupano due caratteri
- I reali in precisione semplice occupano quattro caratteri
- I reali in doppia precisione occupano otto caratteri.

Dopo l'istruzione OPEN dovete dare l'istruzione FIELD per definire l'organizzazione dei campi. La sua sintassi è

FIELD #*numfile, largl* AS *var$1* [,*larg2* AS *var$2*],...

Qui *numfile* indica il numero della periferica, *larg1* è il numero di caratteri nel primo campo e *var$1* è il nome della variabile nel buffer che immagazzinerà il primo campo. Dovete mettere tanti argomenti "*larg* AS *var$*", quanti sono i campi nei record, e la somma delle lunghezze deve essere uguale alla lunghezza del record indicata nell'istruzione OPEN.
Negli esempi di questo paragrafo, i record hanno tre campi: uno per il cognome (lunghezza 25), uno per il nome (lunghezza 15) e uno per il punteggio (reale in precisione semplice). Le istruzioni OPEN e FIELD sono

```
10 REM File ad accesso casuale
20 OPEN "TEST.DAT" AS #1 LEN= 44
30 FIELD #1, 25 AS CG$, 15 AS NM$, 4 AS PT$
```

Per leggere un record usate GET. La sua sintassi è:

GET *#n, numrec*

numrec è il numero del record che viene letto nel buffer. Per esempio per leggere il quarto record date il comando:

```
240 GET #1, 4
```

Per *numrec* potete anche usare una variabile:

```
180 H7 = H8 + 5
190 GET #1, H7
```

Le variabili stringa nel buffer non devono mai essere usate nella parte sinistra di un'istruzione LET: sarebbe un disastro. Inoltre, siccome i numeri sono memorizzati nel file in un formato compatto, prima di poterli leggere dovete usare le funzioni di conversione.
Le funzioni che convertono i dati di un file — dopo che sono stati letti con l'istruzione GET — sono CVI, CVS e CVD rispettivamente per variabili intere, in semplice e in doppia precisione. Le sintassi sono

CVI (*varbuffer$*)
CVS (*varbuffer$*)
CVD (*varbuffer$*)

Per esempio, per stampare il contenuto del buffer nell'esempio precedente, potete usare le istruzioni

```
300 GET #1, RN
310 PRINT CG$; ", "; NM$; ", "; CVS(PT$)
```

La funzione CVS converte la stringa di due caratteri in un numero in precisione semplice. Potete usare le funzioni CVI, CVS e CVD nelle espressioni matematiche:

```
720 TS = CVS(PT$) * 5.3
```

Quando scrivete in un record cancellate tutto quello che era precedentemente immagazzinato in esso, quindi la creazione e l'aggiornamento dei record sono due operazioni simili.
Per scrivere un record in un file ad accesso casuale usate l'istruzione PUT. La sua sintassi è

PUT *#n, numrec*

Analogamente a GET, legge il contenuto del buffer e lo scrive sul disco. Non potete usare LET per assegnare valori alle stringhe del buffer. Per porre le stringhe nel buffer, bisogna usare invece le istruzioni LSET e RSET, simili a LET. LSET pone una stringa nella variabile del buffer allineandola a sinistra, mentre RSET allinea la stringa a destra; LSET è di uso più comune.
Dovete anche essere in grado di convertire i numeri in stringhe per il buffer. Usate le funzioni MKI$, MKS$ e MKD$ che sono esattamente analoghe alle funzioni CVI, CVS e CVD. Le loro sintassi sono:

MKI$ (*intero*)
MKS$ (*precisione semplice*)
MKD$ (*doppia precisione*)

Per esempio

```
100 LINE INPUT "Nuovo cognome    : ";KA$
110 LINE INPUT "Nuovo nome       : ";KB$
120 LINE INPUT "Nuovo punteggio  : "; KC$
130 NP! = VAL(KC$)
140 LSET CG$ = KA$
150 LSET NM$ = KB$
160 LSET PT$ = MKS$(NP!)
180 PUT #1, NR
```

Se vi dimenticate il numero dell'ultimo record letto o scritto, potete usare la funzione LOC la cui sintassi è

LOC (*n*)

Non si usa spesso, dato che la maggior parte dei programmi conserva il numero del record, per le istruzioni PUT e GET, in una variabile numerica.

13.3 Altri dispositivi di I/O

Esaminiamo ora brevemente gli altri dispositivi di input e output per i computer MSX. Per esempio potete accendere e spegnere il vostro mangianastri con MOTOR ON e MOTOR OFF. A prima vista non vi sembrerà molto utile, ma potete costruire altre periferiche, collegarle all'uscita del mangianastri e controllarle con queste istruzioni.
Un'interessante periferica grafica è costituita dalla tavoletta digitale: basta toccare un punto sulla tavoletta e l'MSX-BASIC legge la posizione del punto. La tavoletta digitale è collegata al computer MSX attraverso la porta del joystick.
La funzione PAD legge dati dalla tavoletta. La sua sintassi è:

PAD (*n*)

Qui l'uso di *n*, i cui valori sono riassunti nella Figura 13.5, è alquanto diverso dall'uso che se ne faceva nelle funzioni STICK e STRIG.
Pima di usare PAD(1) o PAD(2) dovete dare un comando PAD(0): il comando PAD(0) controlla se la penna ha toccato la tavoletta, assumendo in caso positivo il valore −1 (vero). Dopo che PAD(0) è risultato vero, PAD(1) assume il valore della coordinata *x* e PAD(2) della coordinata *y*. PAD(3) controlla il pulsante della tavoletta: vale −1 (vero) se il pulsante è pre-

Numero	Significato
0	Legge la tavoletta dalla porta 1 (-1 significa "contatto")
1	Coordinata X già letta dalla porta 1
2	Coordinata Y già letta dalla porta 1
3	Stato del pulsante alla porta 1 (-1 significa "premuto")
4	Legge la tavoletta dalla porta 2 (-1 significa "contatto")
5	Coordinata X già letta dalla porta 1
6	Coordinata Y già letta dalla porta 2
7	Stato del pulsante alla porta 2 (-1 significa "premuto")

Figura 13.5 Valori dell'argomento di PAD

muto e 0 (falso) in caso contrario. Per esempio:

```
130 IF (NOT PAD(0)) THEN GOTO 130
140 PSET (PAD(1), PAD(2))
```

Con un computer MSX è possibile utilizzare anche i paddle, anche se, in verità, si tratta di accessori un po' superati. Si collegano attraverso la porta del joystick e possono essere letti dalla funzione PDL, la cui sintassi è

PDL (*n*)

Come in STICK, *n* è il numero della porta del joystick. La funzione può assumere un valore tra 0 e 255. Potete usare la funzione STRIG per controllare il pulsante sul paddle.

Capitolo

Strumenti di programmazione 14

Questo capitolo presenta molte istruzioni e funzioni che possono essere di grande aiuto nella scrittura dei programmi e nella ricerca degli errori, permettendovi di usare l'MSX-BASIC in modo efficiente.

14.1 Scrittura dei programmi

Oltre ai tasti di controllo del cursore e i tasti INS e DEL che avete già imparato ad usare, ci sono cinque combinazioni di tasti utili per correggere programmi in fase di battitura; li trovate elencati nella Figura 14.1.
L'MSX-BASIC dispone inoltre dell'istruzione AUTO per la numerazione automatica delle linee che fa risparmiare molto tempo nella stesura dei programmi, evitandovi la noiosa battitura dei numeri di linea. La sua sintassi è:

AUTO [*inizio*] [,*incremento*]

Entrambi gli argomenti sono opzionali. L'argomento *inizio* indica il numero di linea da cui deve iniziare la numerazione, mentre *incremento* indica l'intervallo tra i numeri di linea. Il valore di default per entrambi gli argomenti è 10.
Per escludere la numerazione automatica premete CTRL C oppure CTRL STOP. Se il numero di linea mostrato è già stato usato, l'MSX-BASIC pone dopo il numero un asterisco. Per conservare la linea già esistente ed evitare di sostituirla, premete semplicemente il tasto RETURN. Se volete invece sostituire la linea già esistente, basta battere la nuova linea.

Tasti	Funzione
CTRL F	Sposta il cursore all'inizio della parola successiva
CTRL B	Sposta il cursore all'inizio della parola precedente
CTRL N	Sposta il cursore alla fine della linea
CTRL E	Cancella fino alla fine della linea
CTRL U	Cancella l'intera linea

Figura 14.1 Tasti per la correzione di linee di programma

14.2 Gestione degli errori

L'MSX-BASIC offre molti strumenti per trattare gli errori di programmazione. In alcuni casi potete provocare degli errori di proposito per capire cosa sta succedendo nel vostro programma. Finora, quando avete ricevuto un messaggio d'errore, il vostro programma in MSX-BASIC ha semplicemente smesso di girare. Potete far sì che l'MSX-BASIC non si fermi in caso di errore ma salti piuttosto in un'altra parte del programma. L'istruzione ON ERROR GOTO vi permette di avere nel programma degli errori che l'utente non vedrà mai. La sua sintassi è:

ON ERROR GOTO *linea*

linea è il numero della linea di inizio della subroutine di gestione degli errori. Una volta eseguita questa istruzione tutti gli errori faranno saltare l'MSX-BASIC alla linea indicata. Per escludere l'intercettamento degli errori date l'istruzione

ON ERROR GOTO 0

La routine di gestione degli errori può dirvi cosa non ha funzionato per mezzo delle variabili ERR e ERL. ERR contiene il numero dell'errore che è stato commesso ed ERL contiene il numero della linea dove è stato commesso. Ogni errore è contrassegnato da un numero (vedi Appendice B).
La routine di gestione degli errori deve terminare con un'istruzione RESUME che funziona come il RETURN che conclude una subroutine. La

sintassi dell'istruzione RESUME è:

RESUME NEXT
RESUME *linea*
RESUME [0]

RESUME NEXT fa continuare il programma con l'esecuzione della linea che segue quella che ha provocato l'errore (questo fa funzionare l'istruzione ON ERROR GOTO come un'istruzione GOSUB). RESUME con l'argomento *linea* dice all'MSX-BASIC di andare direttamente alla linea indicata. RESUME 0 indica che l'MSX-BASIC deve rieseguire la linea che ha causato l'errore: questo può essere pericoloso a meno che non siate sicuri di aver fatto qualcosa per correggere l'errore.
Eseguite il seguente programma:

```
10 REM Intercettamento di errori
20 ON ERROR GOTO 100
30 PRINT "Commettiamo un errore"
40 SCREEM 2
50 PRINT "E' un errore grave!"
60 END
100 REM Routine di intercettamento degli errori
110 PRINT "E' stato commesso l' errore "; ERR;
" alla linea "; ERL
120 RESUME NEXT
```

Qui si avvisa l'utente che c'è stato un errore nel programma che sta ancora girando. Questo spesso non è utile, dato che la linea non eseguita può essere importante.
Le buone routine di gestione degli errori dovrebbero manipolare gli errori in modo costruttivo. Per esempio, potreste chiedere all'utente di scrivere il nome del file che desidera aprire. Se il file non esiste sul disco, l'MSX-BASIC segnala l'errore numero 53, "File not found". Anziché bloccare il programma potete dare all'utente la possibilità di indicare un nome corretto. Il seguente programma visualizza un file sullo schermo:

```
10 REM Esempio con RESUME 0
20 ON ERROR GOTO 100
30 LINE INPUT "Immetti il nome del file che vuoi
vedere: "; A$
40 OPEN A$ FOR INPUT AS #1
50 IF EOF(1) THEN END
60 LINE INPUT #1, B$
70 PRINT B$
80 GOTO 50
100 REM Routine di gestione dell' errore
```

```
110 LINE INPUT "Il file non e' stato trovato.
Prova con un altro nome: "; C$
120 A$ = C$
130 RESUME 0
```

Quando l'utente immette il nome di un file che non si trova sul disco, la linea 110 avverte di inserire un altro nome e mette questo nome in A$. Quando la linea 40 viene rieseguita, A$ ha un nuovo nome.
L'esempio precedente ha una grossa pecca: contempla solo l'errore di "File not found". Una routine di gestione degli errori dovrebbe essere più efficace di questa, specialmente se si richiede all'utente di immettere dati. Siccome le routine non possono prevedere tutti i casi possibili, potete usare una routine generica che gestisca tutti gli errori possibili che pensate si possano verificare. Sostituite le seguenti linee alle linee dalla 100 alla 130:

```
100 REM Routine di gestione degli errori
110 IF ERR = 53 THEN GOTO 170
120 IF ERR = 56 THEN GOTO 200
130 REM Errore non riconosciuto
140 PRINT "L' MSX-BASIC ha riscontrato l'errore
numero "; ERR; "alla linea "; ERL
150 PRINT "Uscita dall' MSX-BASIC"
160 END
170 LINE INPUT "Il file non e' stato trovato.
Prova con un altro nome: "; C$
180 A$ = C$
190 RESUME 0
200 LINE INPUT "Nome del file non valido.
Prova con un altro nome: "; C$
210 A$ = C$
220 RESUME 0
```

L'MSX-BASIC vi permette di creare dei vostri messaggi d'errore per far scattare le funzioni di intercettamento degli errori. L'istruzione ERROR ha la sintassi:

ERROR *n*

L'argomento *n* è un numero tra 0 e 255; notate che tutti gli errori dal 72 al 255 stampano il messaggio "Unprintable error" a meno che non li intercettiate con ON ERROR GOTO.
Come avete visto, l'istruzione END ferma un programma. L'istruzione STOP è identica a END, tranne che fa stampare all'MSX-BASIC il messaggio "Break in *nn*" dove *nn* è il numero della linea dell'istruzione STOP. Questa non richiede argomenti, perciò la sua sintassi è semplicemente

STOP

L'istruzione STOP può essere utile per controllare le imperfezioni nel programma; dopo aver fermato un programma con STOP, lo potete far ripartire con l'istruzione CONT: l'esecuzione riprende dalla linea successiva a quella indicata nel messaggio "Break in...".
Un metodo più elaborato di ricerca delle imperfezioni in un programma, è costituito dall'uso dell'istruzione TRON che attiva una ricerca passo-passo, cioè visualizza i numeri delle linee nell'ordine in cui vengono eseguite mentre il programma gira normalmente. I numeri di linea vengono stampati fra parentesi quadre:

[*nnn*]

Per disattivare la ricerca, si dà l'istruzione TROFF. Le istruzioni TRON e TROFF sono utili per seguire complicati loop FOR NEXT o programmi pieni di GOTO e GOSUB. TRON e TROFF non richiedono argomenti perciò la sintassi è semplicemente

TRON
TROFF

Per esempio, scrivete il seguente semplice programma:

```
10 REM Esempio dell'uso di TRON
20 PRINT "Ciao"
30 FOR I = 1 TO 3
40 PRINT I
50 NEXT I
```

L'output di questo programma è

```
Ciao
 1
 2
 3
```

Se diamo l'istruzione TRON, l'output diventa:

```
[10][20] Ciao
[30][40] 1
[50][40] 2
[50][40] 3
[50]
```

Se nell'esecuzione di un programma vi appare il messaggio d'errore nu-

mero 7, "Out of memory", vuol dire che il programma è troppo lungo oppure che avete usato stringhe o array troppo grossi. Per controllare la quantità di memoria libera e disponibile in un programma, potete usare la funzione FRE. La sua sintassi è

FRE (*n*)

L'argomento *n* può essere un numero qualsiasi. La funzione FRE restituisce il numero di byte liberi rimasti in memoria. Se questo numero scende al di sotto di 2000 in un grosso programma, esaurirete sicuramente la memoria disponibile in breve tempo. Potreste usare la seguente routine in programmi con problemi di memoria:

```
..........
50 ON ERROR GOTO 1000
..........
400 REM Controlla la memoria
410 IF FRE(0) < 2000 THEN ERROR 100
..........
500 REM Controlla la memoria
510 IF FRE(0) < 2000 THEN ERROR 100
..........
1000 REM Gestione dell' errore
1010 IF ERR = 100 GOTO 2000
.........
2000 PRINT "Alla linea "; ERL; "sono rimasti
solamente "; FRE(0); "byte di memoria"
2010 RESUME NEXT
.........
```

IMPEDIRE INTERRUZIONI DA PARTE DELL'UTENTE

Talvolta bisogna impedire che un programma venga interrotto dall'utente con CTRL STOP. Se per esempio scrivete un programma che apre un file e resta in attesa di dati e l'utente lo interrompe, il file può venir danneggiato. Per evitare questo problema usate le istruzioni STOP ON e ON STOP GOSUB per controllare l'uso di CTRL STOP.
L'istruzione STOP ON ha la sintassi

STOP ON
STOP OFF
STOP STOP

STOP ON dice all'MSX-BASIC di iniziare ad individuare l'eventuale pressione dei tasti CTRL STOP; STOP OFF disattiva questa funzione. STOP STOP sospende momentaneamente il procedimento, ma memorizza se vengono premuti i tasti CTRL STOP in modo che, al successivo comando STOP ON, salta automaticamente alla subroutine.
L'istruzione ON STOP GOSUB dice all'MSX-BASIC quale subroutine eseguire quando si verifica un'interruzione da tastiera.

Capitolo

Altre istruzioni e funzioni 15

Questo capitolo conclude l'esame delle istruzioni e delle funzioni MSX-BASIC; è organizzato secondo l'ordine di importanza degli argomenti.

15.1 Le istruzioni READ e DATA

Con l'istruzione DATA si inseriscono in un programma costanti numeriche e di tipo stringa; queste vengono "lette" e quindi utilizzate dal programma tramite un'istruzione READ, che le assegna a delle variabili. Le istruzioni READ e DATA vengono spesso usate insieme ai loop FOR NEXT per disporre le informazioni negli array.
L'istruzione DATA contiene le informazioni che devono essere lette dall'istruzione READ. La sintassi è

DATA *elemento1, elemento2, ...*

Gli elementi possono essere costanti numeriche o costanti stringa. Le stringhe vanno racchiuse tra virgolette solo se contengono virgole, virgolette o due punti. Le istruzioni DATA possono contenere un numero variabile di elementi a vostro piacimento.
L'istruzione READ legge i dati contenuti nell'istruzione DATA in successione e li memorizza in variabili. La sua sintassi è:

READ *variabile*[,*variabile*]

variabile può essere una variabile di qualsiasi tipo e la lista può essere lunga o corta come volete. Appena l'MSX-BASIC incontra un'istruzione READ, cerca il primo dato non ancora letto in DATA e ne pone il valore nella variabile. Quindi le istruzioni DATA vengono ignorate finché non viene eseguita un'istruzione READ.
Per vedere come funzionano le istruzioni READ e DATA immaginate di voler riempire un array bidimensionale chiamato BP di cinque elementi. Il programma potrebbe essere:

```
..........
30 DIM BP(4, 1)
..........
80 FOR J = 0 TO 4
90 READ BP(J, 0)
100 READ BP(J, 1)
110 NEXT J
..........
430 DATA 12, 43, 7, 90, 16
440 DATA 29, 3
450 DATA 67, 19, 22
..........
```

Notate che le istruzioni DATA hanno una quantità variabile di dati. L'MSX-BASIC cerca semplicemente il dato successivo non ancora letto, per ogni istruzione READ, perciò la lunghezza dell'istruzione DATA è irrilevante.
Il seguente programma svolge la stessa funzione del precedente:

```
..........
30 DIM BP(4, 1)
..........
80 BP (0 ,0) = 12
90 BP (0 ,1) = 43
100 BP (1 ,0) = 7
110 BP (1 ,1) = 90
120 BP (2 ,0) = 16
130 BP (2 ,1) = 29
140 BP (3 ,0) = 3
150 BP (3 ,1) = 67
160 BP (4 ,0) = 19
170 BP (4 ,1) = 22
.......
```

RILETTURA DEI DATI

L'istruzione RESTORE vi permette di riposizionare il puntatore MSX-BASIC e quindi di rileggere i valori contenuti nelle istruzioni DATA. La sua sintassi è

RESTORE [*linea*]

Qui, *linea* è il numero della linea da cui l'MSX-BASIC deve cominciare a leggere i dati. Se viene omessa l'indicazione di *linea*, l'MSX-BASIC inizia a leggere il primo termine della prima istruzione DATA del programma.

15.2 Gestione dei dischetti

L'MSX-BASIC dispone di comandi per la gestione dei file su disco. Questi comandi sono simili a quelli dell'MSX-DOS che vedremo dettagliatamente nel Capitolo 18. Se non avete molta familiarità con l'uso dei dischi, fate quindi riferimento a quel capitolo. Vediamo qui brevemente le principali istruzioni di gestione dei dischetti.
Per formattare un disco, date il comando CALL FORMAT. Vi verrà suggerito di inserire un disco nel drive e vi potrebbe esser chiesto di effettuare alcune scelte. Siccome ogni computer MSX ha un metodo differente di formattazione dei dischi, dovreste leggere la descrizione del comando CALL FORMAT nel manuale del vostro drive.
Per ottenere l'elenco dei file sul disco usate il comando FILES. La sua sintassi è

FILES [*"specfile"*]

Senza argomento il comando visualizza l'elenco di tutti i file presenti sul disco. *specfile* può rappresentare un unico file oppure un gruppo di file attraverso alcuni speciali caratteri jolly (vedi Capitolo 18). Il comando LFILES manda alla stampante l'output del comando FILES.
Per cambiare nome a un file su disco si usa il comando NAME. La sua sintassi è

NAME "*vecchionome*" AS "*nuovonome*"

vecchionome è il nome che il file ha attualmente e *nuovonome* è il nome che volete che assuma.
È ugualmente facile fare una copia di un file su disco con il comando COPY, la cui sintassi è

COPY *file1* TO *file2*

Qui, *file1* è il nome del file che volete copiare e *file2* è il nome da assegnare alla copia del file.
Per cancellare un file usate il comando KILL, la cui sintassi è

KILL "*nomefile*["]

In *nomefile* potete usare i caratteri jolly. Accertatevi di non aver aperto il file che state cercando di cancellare, altrimenti l'MSX-BASIC vi segnala un errore.
Se avete l'MSX-DOS potete passare dall'MSX-BASIC all'MSX-DOS con il comando CALL SYSTEM (o_SYSTEM). Quando fate questo, il programma in memoria viene perduto: è come se spegneste il vostro computer.

15.3 Inizializzazione dei parametri dell'MSX-BASIC

Esistono due istruzioni che inizializzano i parametri generali dell'MSX-BASIC: SCREEN e WIDTH. Abbiamo già visto alcuni degli argomenti di SCREEN, vale a dire il modo e la dimensione degli sprite. Gli altri argomenti controllano parametri che con lo schermo non hanno nulla a che fare.
La sintassi completa di SCREEN è

SCREEN [*modo*] [*dimsprite*] [*clic*] [,*baud*] [,*stamp*]

- L'opzione *clic* accende e spegne il "clic" che sentite alla battuta dei tasti. Per default il clic è attivato (*clic*=1); per spegnerlo ponete *clic*=0.
- L'opzione *baud* dice all'MSX-BASIC se deve usare 1200 o 2400 baud per leggere o registrare sul mangianastri; il valore di default (*baud*=1) è 1200 baud. Per leggere e registrare a 2400 baud ponete *baud* uguale a 2.
- L'opzione *stamp* dice all'MSX-BASIC se la vostra stampante è in grado di stampare tutti i caratteri grafici oppure no. Il valore di default (*stamp*=1) indica che la stampante non stampa caratteri grafici; se la vostra stampante è in grado di farlo ponete *stamp* uguale a 0.

L'istruzione WIDTH controlla la larghezza delle righe dello schermo. Dato che l'MSX-BASIC tende già a scrivere più caratteri possibile, con questa istruzione se ne può solo diminuire il numero. La sintassi è

WIDTH *n*

n è un valore compreso tra 1 e 40.

15.4 Tasti funzione

I tasti funzione (da F1 a F10) sono speciali tasti programmabili, molto comodi poiché riducono la sequenza di tasti da battere. In alcuni computer MSX come l'Hit Bit Sony, i tasti da F6 a F10 non sono inclusi nella tastiera, ma si possono ugualmente ottenere premendo i tasti da F1 a F5 insieme al tasto SHIFT. Normalmente questi tasti sono programmati ad eseguire le istruzioni MSX-BASIC più comuni, ma potete attribuire loro qualsiasi funzione vogliate. Potete programmare ogni tasto inserendovi fino a 16 caratteri di testo: ogni volta che lo premete, per l'MSX-BASIC è come se quei caratteri venissero battuti sulla tastiera. Quindi potete usare i tasti funzione anche per rispondere a degli INPUT o per scrivere i programmi.
L'istruzione KEY elenca le attuali definizioni dei tasti e vi permette di programmare nuove stringhe per ogni tasto. La sintassi è:

KEY LIST
KEY *n*, "*stringa*"

KEY LIST visualizza sullo schermo le definizioni di tutti e dieci i tasti. Nella seconda forma *n* è il numero del tasto funzione, da 1 a 10, mentre *stringa* rappresenta la stringa di caratteri che volete sia prodotta quando premete il tasto.
Per esempio potreste avere un programma nel quale dovete immettere somme in lire. Se dovete scrivere spesso "£000", potete cambiare in questo modo il tasto F1:

```
KEY 1, "£000"
```

Per immettere "£000", dovete premere F1 e il tasto RETURN. Potete evitare anche di battere RETURN modificando il tasto F1 in

```
KEY 1, "£000" + CHR$(13)
```

A prescindere dalla lunghezza della stringa che assegnate a un tasto, nella linea dei messaggi vengono mostrati solo i primi 7 caratteri della stringa. Ricordate che potete eliminare la linea dei messaggi in fondo allo schermo con l'istruzione KEY OFF e riaccenderla con l'istruzione KEY ON.
Potete dire all'MSX-BASIC di andare a una certa subroutine se l'utente preme uno dei tasti funzione, controllando questi ultimi con l'istruzione ON KEY GOSUB e KEY ON. La sintassi di ON KEY GOSUB è

ON KEY GOSUB *linea1*, [*linea2*]...

linea1, linea2... sono i numeri di linea cui deve saltare l'MSX-BASIC quando viene premuto uno dei tasti attivati. L'istruzione KEY ON, la cui sintassi è:

KEY *n* ON

permette di attivare il controllo del tasto funzione corrispondente a *n*. Per esempio, se assegnate al tasto F1 la funzione di "aiuto" nel vostro programma, ogni volta che l'utente preme F1 appare un messaggio che spiega ciò che sta succedendo. Potete assegnare al tasto F2 la funzione "abbandono". Il programma potrebbe assomigliare a questo:

```
..............
130 ON KEY GOSUB 1000, 2000
140 KEY 1 ON
150 KEY 2 ON
..............
190 END
1000 REM Subroutine di aiuto
1010 CLS: PRINT "Menu di aiuto"
.............
1160 REM Fine della subroutine di aiuto
1170 RETURN
2000 REM Subroutine di uscita
2010 CLS: PRINT "Arrivederci"
2020 CLOSE
2030 END
2040 REM Qui il RETURN non c'e' perche'
2050 REM il programma e' finito
```

15.5 Controllo del tempo

Avete già visto come individuare durante l'esecuzione di un programma vari eventi, come la pressione di un tasto o del pulsante del joystick. Per far saltare l'MSX-BASIC a una subroutine, anziché in base a un evento, in base allo scorrere del tempo usate le istruzioni ON INTERVAL GOSUB e INTERVAL ON. Funzionano come le altre istruzioni di intercettamento. La sintassi della prima è

ON INTERVAL= *n* GOSUB *linea*

Il numero *n* rappresenta in cinquantesimi di secondo l'intervallo di tempo tra le successive chiamate della subroutine che inizia in *linea*. Dopo aver dato ON INTERVAL GOSUB, inserite l'istruzione INTERVAL ON

per attivare il controllo (potete disattivarlo con INTERVAL OFF).
Per esempio potreste visualizzare un cronometro in cima allo schermo che indica all'utente quanto tempo gli resta da giocare. Per aggiornarlo ogni 10 secondi il programma può contenere questa routine:

```
300 REM TPR rappresenta il tempo rimasto
310 TPR = 100
320 ON INTERVAL=500 GOSUB 1500
330 INTERVAL ON
340 REM .............
1500 REM Subroutine di stampa
1510 TPR = TPR - 10
1520 XO = POS(O)
1530 YO = CSRLIN
1540 LOCATE 0, 0
1550 PRINT "Tempo rimasto: "; TPR; "secondi"
1560 LOCATE XO, YO
1570 RETURN
```

Siccome l'MSX-BASIC stampa un breve messaggio sullo schermo quando l'evento temporale viene individuato, è importante riportare il cursore alla sua posizione originale prima di lasciare la subroutine.

15.6 Trasferimento dei valori delle variabili

In certi casi può essere necessario scambiare i valori di due variabili. Per fare ciò usate l'istruzione SWAP, che ha la seguente sintassi:

SWAP *var1, var2*

Essa pone il vecchio valore di *var1* in *var2* e viceversa.

15.7 Funzioni definibili

L'MSX-BASIC vi permette con una certa facilità di definire funzioni e di usarle nei programmi. Se potete definire la vostra funzione con una espressione matematica, potete usare l'istruzione DEF FN per creare una funzione che dà valori in base a quella espressione.
La sintassi di DEF FN è

DEF FN *nome* [(*argomento*[,*argomento*]...)]=*espressione*

nome deve essere un nome di variabile valido che diventa, preceduto da FN, il nome della funzione; *argomento* è un nome di variabile che compare in *espressione*, che deve essere sostituito da un valore ogni volta che la funzione viene richiamata; *espressione* è una qualsiasi espressione MSX-BASIC. Per esempio, potreste definire una funzione che calcola il seno iperbolico di un numero. Per definirla usate

```
30 DEF FNHS(X) = (EXP(X) - EXP(-X))/2
```

La X non è importante; tutto quello di cui si ricorderà l'MSX-BASIC è che la funzione FNHS usa un argomento reale in doppia precisione. Potete ora usare la funzione FNHS come una normale funzione MSX-BASIC:

```
90 G = FNHS(3.1)
```

La principale limitazione dell'istruzione DEF FN è che dovete definire la vostra funzione con una sola espressione, il che è accettabile per funzioni semplici, ma di solito vi impedisce di definire funzioni complesse.

15.8 L'istruzione CALL

Mediante CALL potete accedere a programmi speciali scritti nella ROM. Questi programmi possono variare da computer a computer. La sintassi dell'istruzione CALL è

CALL *routine* (*param1, param2,...*)

routine è il nome del programma nella ROM; se sono necessari dei parametri li dovete indicare tra parentesi. Invece della parola CALL potete anche usare il trattino di sottolineatura (_):

_*routine* (*param1, param2,...*)

Per esempio lo Yamaha CX5M contiene un sintetizzatore musicale che si può controllare con il programma MUSIC. Per far eseguire il programma MUSIC dall'MSX-BASIC, si deve dare l'istruzione

```
CALL MUSIC
```

oppure

```
_MUSIC
```

15.9 Fusione di programmi

In MSX-BASIC è estremamente semplice fondere due programmi in uno; è comunque un'operazione da fare con estrema attenzione per evitare sovrapposizioni di linee con lo stesso numero. Il comando che serve a fondere due programmi è MERGE; la sua sintassi è

MERGE "*nomefile*"

Questo carica il file indicato nella memoria e lo riunisce al programma già residente. Il file deve però essere stato salvato in formato ASCII, cioè aggiungendo l'opzione A a SAVE:

SAVE "*nomefile*", A

15.10 Cancellazione della memoria

Durante l'esecuzione di un complicato programma MSX-BASIC potreste finire per non avere abbastanza memoria. L'MSX-BASIC vi dà il modo di gestire la memoria libera disponibile attraverso il comando CLEAR e il comando ERASE.
CLEAR azzera tutte le variabili in memoria e vi permette di predisporre lo spazio da riservare alle stringhe. Questo, ovviamente, è utile soltanto all'inizio di un programma, prima che qualsiasi variabile sia stata inizializzata. La struttura è

CLEAR [*dimstringhe*]

Il valore di default di *dimstringhe*, che rappresenta la quantità di RAM disponibile per le variabili stringa, è di 200 byte. Il comando CLEAR inoltre, chiude tutti i file.
Il comando ERASE libera lo spazio usato da un array e fa compattare la memoria all'MSX-BASIC. La sintassi del comando è

ERASE *nomearray*[,*nomearray*]...

nomearray è il nome dell'array che volete cancellare dalla memoria. Dopo il comando ERASE potete nuovamente dimensionare l'array con il comando DIM.

Capitolo

Programmare in Assembler 16

Quando programmate in MSX-BASIC, un altro programma, chiamato interprete BASIC, traduce i vostri comandi nell'Assembler Z80, linguaggio che può essere capito dalla CPU (anch'essa Z80). La maggior parte delle persone non si preoccupa di imparare l'Assembler: non è semplice e non è utile per buona parte degli utilizzatori di un computer MSX.
Potrebbe però attirarvi la prospettiva di lavorare in un linguaggio che può eseguire parecchie migliaia di istruzioni al secondo sul vostro computer MSX (da 100 a 300 volte più velocemente dell'MSX-BASIC). Potete scrivere programmi in Assembler, inserirli in un programma in MSX-BASIC e farli girare insieme ad esso; potete programmare lavori complessi in Assembler Z80 e lavori più semplici in MSX-BASIC.
Potete, inoltre, usare l'MSX-BASIC per leggere e modificare alcune locazioni nella RAM che non sono accessibili ai comandi e alle funzioni solitamente utilizzati. La conoscenza della logica dell'Assembler vi permette quindi di svolgere compiti in MSX-BASIC che sono normalmente riservati ai programmi in linguaggio Assembler.

16.1 Accedere alla RAM

Molte locazioni nella RAM contengono valori importanti ai fini dei vostri programmi. Per esempio, anche se potete predisporre molti valori in MSX-BASIC, non esistono poi comandi che controllino quei valori più avanti nel programma. Per scoprire cosa c'è in una certa locazione di memoria, dovete usare la funzione PEEK.

La sintassi della funzione PEEK è:

PEEK (*ind*)

L'argomento *ind* è l'indirizzo o locazione del byte della RAM che volete controllare; può andare da 0 a 65535. Un indirizzo indica un certo numero di byte dall'inizio della RAM e la funzione PEEK può leggere ogni byte della RAM, sia che faccia parte di un programma in MSX-BASIC, sia dell'interprete BASIC, sia della zona della RAM che contiene i valori predisposti dall'MSX-BASIC a proprio uso.
Notate che gli indirizzi da 0 a 32767 sono in realtà nella ROM e non nella RAM, comunque con la funzione PEEK potete leggere anche queste locazioni come se facessero parte della RAM.
Per esempio, mentre vi trovate in modo 2, potreste voler trovare le coordinate del cursore grafico che non possono essere lette da alcuna funzione MSX-BASIC. Questi valori sono conservati nella RAM, ad uso dell'MSX-BASIC, agli indirizzi &HFCB3 e &HFCB5 (gli indirizzi sono normalmente espressi in esadecimale). Quindi potete trovare i valori con i comandi

```
GX = PEEK(&HFCB3)
GY = PEEK(&HFCB5)
```

L'uso incauto della funzione PEEK non può in ogni caso provocare danni. Se vi sentite in vena di avventure, potete modificare le locazioni di memoria con l'istruzione POKE. Siate avvisati, comunque, che modificare zone poco familiari della RAM può causare risultati imprevisti, come perdere un programma MSX-BASIC residente in memoria. Un uso poco avveduto dell'istruzione POKE può anche pregiudicare il funzionamento del vostro computer MSX e non vi rimarrà che spegnerlo, con la relativa perdita di dati e programmi. In altre parole, state attenti.
La sintassi di POKE è

POKE *ind, valore*

L'argomento *ind* è lo stesso della funzione PEEK; *valore* è un numero tra 0 e 255 da porre nella locazione. Notate che è inutile usare POKE con un indirizzo inferiore a 32768, poiché questa area è occupata dalla ROM che è inaccessibile per l'istruzione POKE.
A parte il caricamento di routine in Assembler nella RAM non ci sono molti usi possibili per l'istruzione POKE. Per vedere come funziona questa istruzione, modificate la larghezza dallo schermo in modo 0 senza cancellarne il contenuto. L'indirizzo &HF3B0 contiene la lunghezza delle linee; per portarlo a 34 date il comando:

```
POKE &HF3B0, 34
```

L'Appendice E elenca alcune aree di memoria interessanti dove potete usare la funzione PEEK o l'istruzione POKE. Notate che molte di queste locazioni sono accessibili per la sola lettura; modificare il valore di una locazione con l'istruzione POKE a volte, anche se non sempre, modifica anche il funzionamento dell'MSX-BASIC.

16.2 Eseguire programmi in Assembler

Se sapete programmare nell'Assembler Z80 potete includere programmi in questo linguaggio all'interno dei programmi in MSX-BASIC. Per far questo seguite questi passi:

- Caricate il programma in memoria o nelle variabili MSX-BASIC
- Dite all'MSX-BASIC dove si trova il programma
- Dite all'MSX-BASIC di eseguire il programma della RAM

Questi passi sono piuttosto semplici, ma dovete prima caricare in memoria il vostro programma. Il metodo più comune è porre i passi del programma in un array di interi. Di solito l'array viene riempito con numeri ordinati in un'istruzione DATA. Per esempio:

```
200 DIM PG%(1000)
210 FOR I = 0 TO 1000 STEP 2
220 READ A, B
230 PG%(I) = 256*A + B
240 NEXT I
1000 DATA 12, 210, 48
```

Potete anche usare le istruzioni BLOAD e BSAVE per salvare e rileggere i dati dalla memoria (vedi paragrafo 16.5). Il programma in entrambi i casi deve comunque terminare con l'istruzione RET del linguaggio Z80.
Una volta che il vostro programma è nella memoria, dovete indicare dove inizia con l'istruzione DEFUSR, che ha la seguente sintassi:

DEFUSR [*n*] = *ind*

L'argomento *n* va da 0 a 9 e si possono quindi definire fino a 10 programmi. Se tralasciate *n* l'MSX-BASIC presume che vi riferiate all'USR0. L'argomento *ind* è l'indirizzo da cui inizia il programma. Quando usate BLOAD, *ind* è semplicemente l'inizio del programma.
L'uso di DEFUSR insieme a un array di interi è un po' più complicato.

Per determinare l'indirizzo iniziale dell'array dovete usare la funzione VARPTR. La cui sintassi è

VARPTR (*variabile*)

e che dà come risultato l'indirizzo della *variabile*. Per usarla con l'esempio precedente date l'istruzione

```
250 DEFUSR0 = VARPTR(PG%(0))
```

Per far partire un programma in Assembler usate la funzione USR la cui sintassi è

var = USR [*n*] (*arg*)

Come nell'istruzione DEFUSR *n* va da 0 a 9 oppure, se viene tralasciata, si presume uguale a 0. La variabile *var* è il risultato del programma in Assembler, mentre *arg* è la variabile trasmessa al programma. Queste due variabili devono essere presenti, ma possono essere variabili fittizie, nel senso che potete assegnare loro un nome qualsiasi.
Se passate un argomento *var* alla funzione USR, il vostro programma in Assembler può controllare il registro A per riconoscere il tipo della variabile. Se usate una variabile stringa (il registro A, in questo caso, vale 3), il registro DE punta al puntatore da tre byte della stringa (lunghezza e indirizzo di inizio dei dati). Se usate una variabile numerica, il registro HL punta al numero assunto dalla variabile in quel momento. Per riportare il valore a quello di *arg* riordinate i registri come qui descritto.
Per eseguire la routine in Assembler dell'esempio precedente, usate l'istruzione

```
260 DM = USR0(DM)
```

Se immagazzinate il programma in Assembler in una variabile, siate sicuri di richiamare il programma subito dopo aver determinato l'indirizzo della variabile, in modo che l'MSX-BASIC non cambi l'indirizzo in un eventuale rimescolamento della memoria.
L'Appendice E elenca alcuni puntatori nella ROM dell'MSX-BASIC che potete richiamare con la funzione USR. Queste locazioni non richiedono argomenti. Per esempio, l'unico modo per riordinare rapidamente i registri sonori dell'MSX-BASIC è dare l'istruzione BEEP. Se volete farlo, però, senza provocare il "beep", battete le istruzioni

```
DEFUSR = &H0090
DM = USR(DM)
```

16.3 Porte di input/output

Se volete realizzare degli input o degli output direttamente in Z80, potete usare la funzione INP e il comando OUT dell'MSX-BASIC, con le seguenti sintassi:

INP (*porta*)
OUT *porta, dato*

L'argomento *porta* rappresenta l'indirizzo della porta e *dato* è un valore che può andare da 0 a 255. L'Appendice E contiene un elenco delle porte disponibili per la programmazione.
Il comando WAIT attende che i dati provenienti da una porta soddisfino determinate condizioni; è usato raramente. La sua sintassi è

WAIT *porta, esprand, esprxor*

L'MSX-BASIC legge qual è la *porta* da dove devono giungere i dati; trova i dati, li confronta con *esprxor* che è una istruzione XOR, poi confronta il risultato con *esprand* che è una istruzione AND. Se il risultato è 0, l'MSX-BASIC continua ad aspettare.

16.4 Gestione avanzata del video

Alcune istruzioni e funzioni MSX agiscono sulla RAM controllata dal video processore TI 9918A. Il loro uso influisce sulla normale visualizzazione e richiede una approfondita conoscenza della struttura del VDP e del contenuto dei registri.
La funzione VDP vi permette di leggere e modificare i registri del chip VDP. Per modificare un registro si usa:

VDP (*reg*)=*valore*

reg è il numero del registro che può variare da 0 a 7; *valore* va da 0 a 255. Per leggere un registro usate la funzione

VDP (*reg*)

Siccome ogni bit del registro è significativo, accertatevi di comprendere i valori nei registri prima di modificarli.
La funzione VPEEK e l'istruzione VPOKE sono molto simili a PEEK e

POKE tranne che essi operano su 16 K di RAM del video. Questo è il solo modo per accedere alla RAM del video, che contiene sia i valori per pixel e caratteri sullo schermo, sia i dati sugli sprite e sui colori usati. Gli indirizzi della VRAM vanno da 0 a 16383.
La funzione BASE permette di cambiare l'indirizzo base usato dal chip VDP quando mostra testo e grafica sullo schermo. Serve solo ai programmatori esperti che vogliono memorizzare nella VRAM più immagini. Per predisporre un registro usate:

BASE (*n*)=*indvram*

I valori di *n* sono mostrati nella Figura 16.1; *indvram* è l'indirizzo della VRAM a cui volete che punti l'elemento. Per leggere il valore corrente della tabella usate:

BASE (*n*)

Per usare la funzione BASE, dovete indicare il modo del video, moltiplicarlo per 5 e aggiungere il risultato al parametro mostrato in Figura 16.1. Quindi per porre in C2 l'indirizzo di base delle caratteristiche di uno sprite al livello 2, usate il comando:

```
C2 = BASE((2*5)+3)
```

Parametro	Significato
0	Indirizzo base per la tabella dei nomi
1	Indirizzo base per la tabella dei colori
2	Indirizzo base per la tabella generatrice di forme
3	Indirizzo base per la tabella degli attributi degli sprite
4	Indirizzo base per la tabella delle forme degli sprite

Figura 16.1 Parametri della funzione BASE

16.5 Memorizzazione di immagini binarie

Potete caricare o salvare una copia della RAM su disco o cassetta con i comandi BLOAD e BSAVE. Questi comandi servono quasi esclusivamente a salvare programmi in linguaggio Assembler o a salvare una copia di parte della VRAM.
La sintassi del comando BSAVE è

BSAVE "*nomefile*", *indiniz*, *indfin*

nomefile è il nome del file nel quale volete immagazzinare l'immagine binaria (potrebbe essere CAS: se lo volete salvare su cassetta). *indiniz* e *indfin* sono gli indirizzi iniziale e finale della parte di RAM che volete salvare. Questi due indirizzi vengono memorizzati nel file.
La sintassi del comando BLOAD è

BLOAD *nomefile* [,*offset*]

Se includete l'indirizzo di offset, esso viene aggiunto agli indirizzi iniziale e finale, memorizzati nel file; questo vi permette di caricare un blocco di memoria in un punto differente da quello da cui è stato salvato.
Se volete salvare un'immagine della VRAM dovete usare l'opzione S. L'opzione R manda in esecuzione il programma Assembler caricato. Le sintassi allora diventano

BSAVE "*nomefile*", *indiniz*, *indfin*, S
BLOAD "*nomefile*" [,S] [,R] [,*offset*]

Parte Terza

L'MSX-DOS

Capitolo

Introduzione all'MSX-DOS

Se avete un drive e 64 K di RAM, potete usare l'MSX-DOS, che è il sistema operativo su disco usato dai computer MSX. L'MSX-DOS è necessario se volete utilizzare dei programmi applicativi al di fuori dell'MSX-BASIC. Se però volete usare il disco solamente con l'MSX-BASIC, non avete bisogno di usare l'MSX-DOS.
Molti programmi applicativi per computer MSX richiedono l'utilizzo dei comandi DOS e ne presuppongono la conoscenza. Alcune operazioni (come la formattazione di un dischetto) non vengono eseguite dai programmi applicativi, ma devono essere svolte per mezzo dell'MSX-DOS o, in certi casi, con i comandi per disco dell'MSX-BASIC.

17.1 Cosa fa l'MSX-DOS

Potete immaginare che un sistema operativo sia come il quadro di comando di un aeroplano: senza di esso l'aeroplano non potrebbe volare. Il sistema operativo fa sì che l'utente (il pilota) possa controllare il computer dicendogli dove andare e cosa fare. Il sistema operativo coordina le parti del computer e vi fornisce un semplice modo per controllarle. In questo capitolo inizierete a capire come l'MSX-DOS svolge questi compiti.
L'MSX-DOS è un'estensione del sistema operativo MSX; permette a quest'ultimo di controllare i drive. Comunque l'MSX-DOS fa molto più di questo: fornisce un modo per dire al computer quale programma o comando deve eseguire, dove può trovare il programma o il comando e cosa deve fare con esso.

Il principale livello su cui opera l'MSX-DOS è rappresentato dalla cosiddetta funzione di utility: in questo modo l'MSX-DOS esegue comandi che vi permettono di interagire direttamente con il vostro computer. Questi comandi svolgono funzioni come assegnare nomi ai file su disco oppure copiare file da un disco all'altro.
L'MSX-DOS considera i propri comandi come se fossero programmi applicativi. Questi, tuttavia, sono più limitati rispetto ai programmi applicativi veri e propri, non svolgono funzioni di word processor o di gestione; sono invece usati per la gestione e il controllo del computer. Ogni comando ha un nome che solitamente è facile da ricordare. Per esempio, per copiare un file da un disco a un altro si usa il comando COPY. Tutti questi comandi vengono descritti nel Capitolo 19.

17.2 Uno sguardo all'interno dell'MSX-DOS

Molte persone utilizzano l'MSX-DOS senza sapere esattamente cosa fa il sistema. La comprensione, anche minima, di come funziona l'MSX-DOS, può aiutarvi ad usare in modo efficace il vostro sistema operativo e a farvi comprendere i limiti delle funzioni che l'MSX-DOS può svolgere.
Se poteste vedere all'interno dell'MSX-DOS, vedreste una massa molto complicata di istruzioni in Assembler. Per fortuna, non avete bisogno di conoscere questo linguaggio per usare l'MSX-DOS né avete bisogno di sapere come l'MSX-DOS svolge il suo lavoro. Tuttavia non è difficile capire il processo mediante il quale l'MSX-DOS esegue le funzioni ed è utile saperne qualcosa, specialmente quando si impartiscono comandi direttamente all'MSX-DOS.

COME L'MSX-DOS ESEGUE I COMANDI

I processi esaminati in questa parte si riferiscono alle funzioni di utility dell'MSX-DOS, di cui imparerete il funzionamento nei prossimi due capitoli. L'MSX-DOS è come un programma che lavora in continuazione. Quando accendete il computer, l'MSX-DOS viene letto dal disco, posto nella RAM ed attivato. Quando l'MSX-DOS è pronto a ricevere un comando o a eseguire un programma, stampa un messaggio sullo schermo e attende le vostre istruzioni.
Questo messaggio, solitamente "A>", avverte che l'interprete di comandi dell'MSX-DOS è pronto per la prossima istruzione. L'interprete legge il vostro comando, trova il programma necessario e inizia ad eseguirlo. (Anche se in seguito verrà usata la parola "comando", il procedimento è ugualmente applicabile a qualsiasi programma eseguito dall'MSX-DOS.)

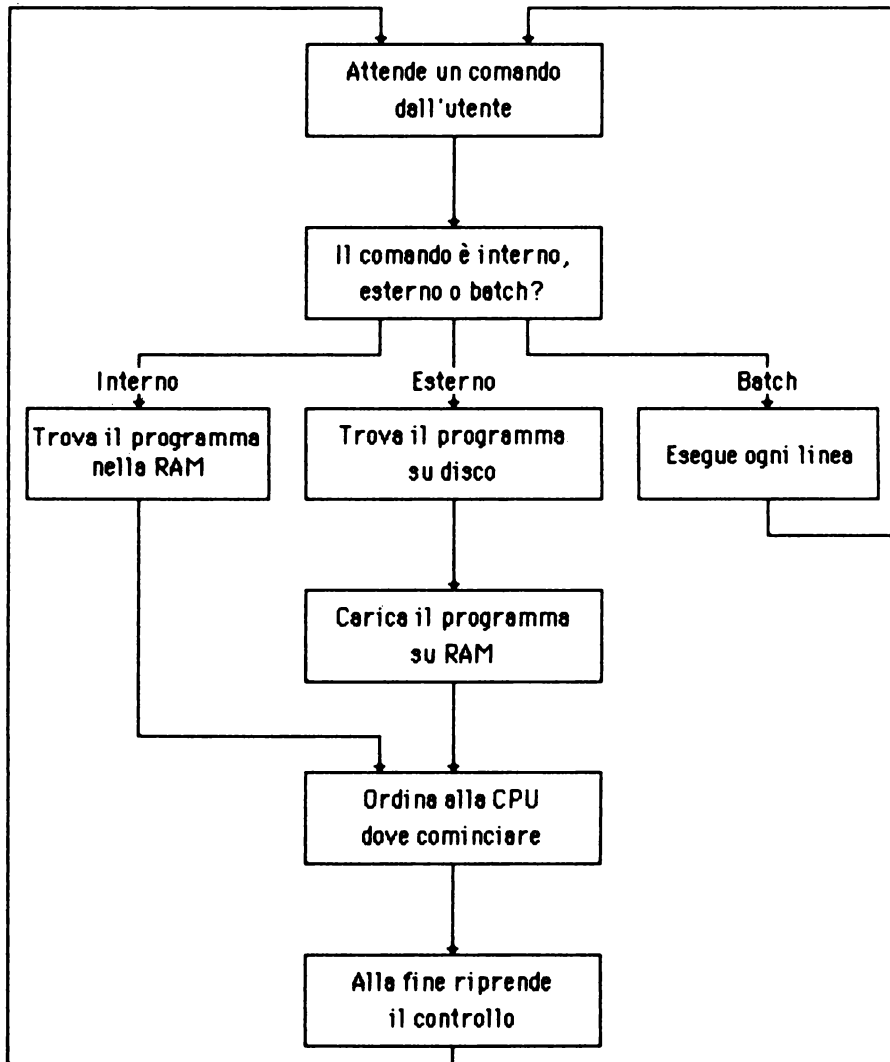



Figura 17.1 Come l'MSX-DOS esegue un programma

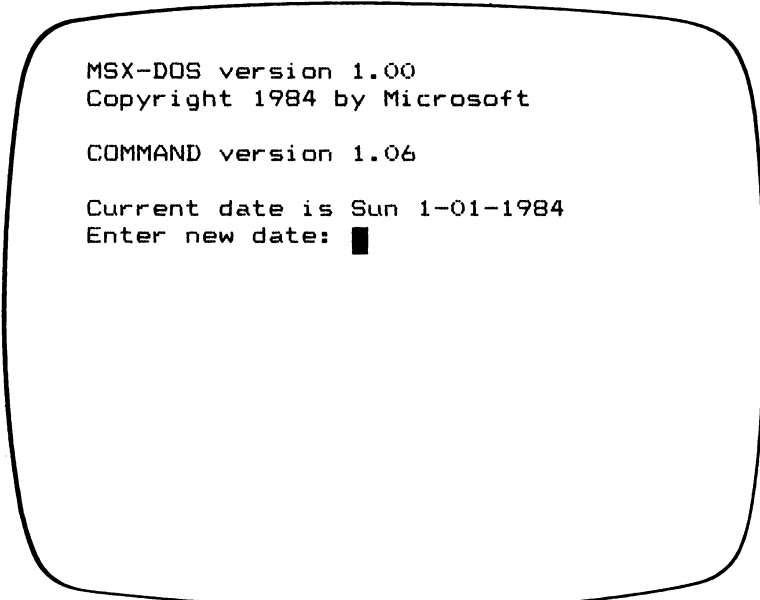
Per eseguire un comando dovete semplicemente batterne il nome (e gli eventuali argomenti) sulla tastiera. L'MSX-DOS mostra i caratteri sullo schermo mentre li battete. Poi premete il tasto RETURN, come si fa con l'MSX-BASIC.

Dopo che avete fornito all'MSX-DOS il comando, il sistema operativo deve trovare il programma che si riferisce al comando da eseguire. Ci sono due possibilità: il comando si può trovare nella memoria interna od esterna. I comandi interni si trovano nell'MSX-DOS stesso. Ciò significa che l'MSX-DOS non deve cercarli sul disco, poiché sono caricati nella RAM, insieme all'MSX-DOS. I programmi applicativi invece sono esterni. Ogni volta che riceve questi comandi, l'MSX-DOS deve leggerli dal disco prima di eseguirli (per esempio un programma di word processing non residente nella ROM).
Un *file batch* è un tipo speciale di comando costituito da un insieme di comandi MSX-DOS. La Figura 17.1 mostra i passi seguiti dal sistema operativo quando esegue un comando.
Non c'è differenza nella chiamata di un comando esterno o interno e spesso non vi interessa conoscerne la distinzione; quando eseguite un comando esterno però dovete essere sicuri che nel drive ci sia il disco che contiene quel comando, perché in caso contrario, l'MSX-DOS dà il messaggio d'errore "Bad command or file name".

Capitolo

Introduzione ai comandi e ai file 18

L'apprendimento dell'MSX-DOS risulta più facile dopo la comprensione di alcuni concetti base; in questo capitolo verrà mostrato:

- Come usare i comandi MSX-DOS ed eseguire programmi applicativi
- Come usare i file
- Come scrivere i file batch

Inoltre, in questo capitolo cominceremo a conoscere alcuni comandi MSX-DOS che svolgono alcuni tra i compiti più comuni. Per informazioni più approfondite e per l'esame di tutti i comandi rimandiamo al Capitolo 19 che potete usare anche come guida di riferimento.
Buona parte dei comandi presentati in questo capitolo è accompagnata da esempi. Provate i comandi sul vostro computer MSX mano a mano che vengono presentati. La funzione di ogni comando dovrebbe essere facile da ricordare poiché la maggior parte dei comandi MSX-DOS ha un nome strettamente legato all'azione che svolge.

18.1 Avviamento dell'MSX-DOS

Seguite le istruzioni allegate al vostro drive per avviare l'MSX-DOS. Accertatevi che la cartuccia del drive sia ben inserita nel connettore (a meno che il vostro computer MSX non abbia la cartuccia all'interno). Se il drive ha un interruttore, accendetelo e poi accendete il vostro computer. Caricato l'MSX-DOS, la prima cosa che vedete è, probabilmente, la scrit-

ta con il numero della versione dell'MSX-DOS e il copyright Microsoft, seguiti dal messaggio di richiesta della data: "Enter new date". Questo accade a meno che il vostro computer MSX non abbia un orologio interno oppure il disco di sistema non contenga un file speciale chiamato AUTOEXEC.BAT che, quando attivate il sistema, esegue automaticamente una serie di comandi scritti al suo interno. I file AUTOEXEC.BAT sono trattati per intero alla fine di questo capitolo. La richiesta della data si presenta in questo modo:

```
MSX-DOS version 1.00
Copyright 1984 by Microsoft

COMMAND version 1.06

Current date is Sun 1-01-1984
Enter new date: █
```

Il quadratino lampeggiante che appare alla destra del messaggio è detto cursore, come nell'MSX-BASIC. Notate che il cursore è in attesa dopo i due punti alla seconda riga.
La data che batterete deve essere costituita da tre numeri separati da trattini (–) o da barre (/). Ad esempio, per il 9 novembre 1985:

9-11-85
9/11/85
09/11/1985

Se commettete un errore mentre state scrivendo, premete il tasto BACKSPACE e riscrivete i caratteri sbagliati. Quando avete terminato premete il tasto RETURN.
L'MSX-DOS vi chiede la data perché, quando salva un file su disco, registra la data in cui quel file è stato creato. La registrazione della data non

è, comunque, indispensabile, anche se è una buona abitudine da prendere. Se non vi interessa battere la data corretta, premete semplicemente il tasto RETURN e l'MSX-DOS assumerà come data il 1 gennaio 1984 (o qualsiasi data sia scritta). Dopo che avete immesso la data, appare il prompt:

```
A>█
```

Il prompt standard A> è il modo che l'MSX-BASIC usa per dirvi che sta aspettando un comando da voi e che nessun altro programma è in fase di esecuzione.

18.2 Significato del prompt

Ora che A> è sullo schermo il passo successivo è dare un comando all'MSX-DOS. In genere, per dare un comando e far girare un programma, dovete battere il nome del comando o del programma e premere il tasto RETURN. Vedremo nel corso del capitolo che il nome del comando può essere seguito da altre specifiche. Un primo comando utile può essere il comando DIR (*directory*=catalogo), che vi mostra sullo schermo la lista di tutti i file che si trovano su un disco. Ecco un esempio in cui viene utilizzato questo comando.

> Quando compare il messaggio A>, scrivete DIR (in caratteri maiuscoli o minuscoli):
>
> ```
> A>DIR
> ```
>
> Premete ora RETURN per far eseguire il comando: otterrete la lista di tutti i file presenti sul drive A:. Avete appena dato il vostro primo comando MSX-DOS.

Quando battete i comandi potete usare alcuni tasti speciali: potete premere il tasto BACKSPACE per cancellare un carattere sbagliato (come nell'MSX-BASIC), mentre per cancellare un'intera linea potete premere il tasto ESC O ESCAPE.
Se l'MSX-DOS vi manda il messaggio d'errore "Bad command or file name" quando cercate di far eseguire un comando, non dovete preoccuparvi. Si tratta dell'errore più comune, dovuto ad un errore di battitura del nome del programma.

IL DRIVE DI DEFAULT

Come avete visto, il prompt dell'MSX-DOS vi dice che il sistema operativo è in attesa di un comando e vi indica contemporaneamente in quale drive l'MSX-DOS cercherà i comandi o i file. Questo drive è detto *drive di default*: esso cioè viene scelto se non ci sono specifiche contrarie.
La maggior parte dei computer MSX ha un solo drive, perciò potreste anche non aver bisogno di preoccuparvi di come si usa un secondo drive. Anche se avete un solo drive, in una delle sezioni seguenti imparerete come far credere all'MSX-DOS di averne due.
Quando appare il prompt A>, il drive di default è il drive A:. Quando appare il prompt B>, l'MSX-DOS indica che il drive di default è il B: (notate che le lettere che indicano i drive sono seguite dai due punti per distinguerle dai nomi dei file).
Quando date un comando come DIR, non importa sapere con quale drive è collegato l'MSX-DOS; è importante invece saperlo in caso di comandi esterni, poiché l'MSX-DOS tenta di trovare il comando sul drive di default.
Quando dite all'MSX-DOS di eseguire un comando esterno, la ricerca, salvo diverse indicazioni da parte vostra, avviene solo sul disco nel drive di default. Se i comandi che volete eseguire sono sul drive B:, dovete cambiare il drive di default da A: in B:. Per fare ciò basta battere la nuova lettera seguita dai due punti e premere RETURN.

Per esempio, per cambiare il drive di default da A: in B:, date il comando

```
A>B:
B>
```

Il prompt B> mostra che il drive di default è ora B:.

Se avete un solo drive, l'MSX-DOS lo denomina A:. Se voi, però, indicate il drive B:, l'MSX-DOS vi chiede di inserire un nuovo disco. Per esempio, se il drive corrente è A: e voi richiedete un file sul drive B:, appare il messaggio:

```
Insert diskette for drive B:
and strike a key when ready
```

Estraete il disco dal drive e inserite quello cui volete accedere e premete un qualsiasi tasto. L'MSX-DOS ora considera il drive indicandolo con la lettera che avete specificato.
Se il programma non è nel drive di default e non avete indicato il disco su cui guardare, l'MSX-DOS visualizza sullo schermo il messaggio d'errore "Bad command or file name".

Cambiare il drive di default è quindi un modo per eseguire comandi da un disco diverso, ma questo procedimento spesso è noioso. Fortunatamente esiste un modo più semplice per dire all'MSX-DOS dove deve cercare i programmi: indicate semplicemente il nome del drive prima di quello del programma. Per esempio:

```
B>A: CONTAB
```

18.3 Come fornire argomenti ai comandi

I comandi MSX-DOS che abbiamo esaminato finora sono abbastanza semplici. È possibile fornire più informazioni all'MSX-DOS attraverso i comandi, in modo che possa svolgere compiti più complessi.
Per fornire queste informazioni si usano gli argomenti, come abbiamo visto anche nei comandi MSX-BASIC. Quasi tutti i comandi richiedono o ammettono argomenti e il loro uso vi permette di utilizzare il computer con più efficacia.

Per esempio, avete già visto che il comando DIR stampa sul video l'elenco dei file sul disco:

```
A>DIR
```

Il comando DIR senza argomenti elenca i file presenti sul drive di default, in questo caso A:. Potete fornire al comando un argomento che specifica qual è il drive di cui volete vedere il directory. Per esempio, per vedere i file sul drive B:, scrivete:

```
A>DIR B:
```

Qui B: è un argomento che spiega al comando DIR, di visualizzare i file sul drive B: e non quelli sul drive di default.

Il comando DIR può funzionare con o senza argomento. Alcuni comandi, invece, come nell'MSX-BASIC, hanno bisogno sempre di argomenti in quanto devono sapere su cosa devono lavorare.

Per esempio, il comando TYPE, che visualizza sullo schermo il contenuto di un file, richiede che gli indichiate il nome del file. Supponete di voler vedere il testo di un file chiamato COGE.RPT. Potete dare il comando

```
A>TYPE COGE.RPT
```

e il contenuto del file COGE.RPT viene stampato sul video. Se battete solamente il comando TYPE senza argomenti, l'MSX-DOS risponde con il messaggio d'errore "File not found".

Il comando TYPE non è in grado di visualizzare tutti i file in forma leggibile. Se cercate di stampare un file non scritto con il set di caratteri ASCII, verrà visualizzata una serie di caratteri illeggibili.
Vedremo che sono molti i tipi di argomenti che potete fornire ai comandi; i più comuni sono i nomi dei drive e quelli dei file che abbiamo usato nei due esempi precedenti. Troverete altri tipi di argomenti nel Capitolo 19.

18.4 Alcuni semplici comandi: FORMAT e COPY

Ora che avete visto come far partire l'MSX-DOS e come dargli i comandi, siete pronti per impiegare il computer per usi più pratici. La prima cosa da imparare è come inizializzare i dischi e come farne delle copie.

FORMATTAZIONE DEI DISCHI

I dischi appena acquistati, generalmente, non sono pronti per l'uso. Prima di tutto vanno formattati. Il procedimento di formattazione consiste nel porre sul disco dei punti di riferimento (tracce e settori) che indichino all'MSX-DOS dove mettere i dati sul disco stesso. Tenete presente, comunque, che tutto ciò che si trova sul disco prima della formattazione viene perduto in ogni caso.
Formattare un disco è un po' come dipingere strisce bianche in un parcheggio: come le strisce indicano dove parcheggiare le automobili, così la formattazione di un disco indica all'MSX-DOS dove mettere i dati. Se l'area di parcheggio è nuova, non ci saranno precedenti strisce; se invece ci sono, prima di tracciare le nuove strisce, le vecchie dovranno essere cancellate. Formattando il disco, tutte le precedenti informazioni vanno perdute.
Perciò, quando usate il comando FORMAT, dovete stare molto attenti a non usare un disco che contenga informazioni che vi possono ancora servire. Controllate il contenuto di un disco con il comando DIR, prima di formattarlo.
Il comando FORMAT non richiede argomenti, se il drive è uno solo. Se invece il disco da formattare è inserito in un drive che non è quello di default, bisogna specificarlo nel comando.

Per esempio, per formattare un dischetto nel drive B:, date il comando

```
A>FORMAT B:
```

Quando l'MSX-DOS vi chiede in quale drive è inserito il disco da formattare, scrivete B:. Quando vi dice di inserire il disco nel drive B:, eseguite e premete un tasto qualsiasi. Tutto questo formatterà il dischetto nel drive B:.

COPIATURA DI FILE

Il comando COPY vi permette di copiare i file da un disco a un altro. Questo comando deve avere almeno due argomenti: il nome del file che volete copiare e il nome del drive su cui lo volete copiare. (In realtà le prestazioni di questo comando vanno ben oltre, come potrete vedere nel Capitolo 19.) Il comando COPY dell'MSX-DOS è simile al COPY dell'MSX-BASIC.

Per esempio se date il comando

```
A>COPY COGE.DAT B:
```

l'MSX-DOS copia il file COGE.DAT dal disco nel drive A: al disco nel drive B:.

RIPRODUZIONE DEL DISCO DI SISTEMA

Ora che sapete usare i comandi FORMAT e COPY, potete impiegarli in modo più efficiente. In questo paragrafo imparerete a fare una copia di riserva, detta *backup*, del disco di sistema dell'MSX-DOS.
La ragione per cui è consigliabile fare una copia del disco di sistema è che non conviene fare affidamento su un unico disco per le necessità di ogni giorno perché gli incidenti sono frequenti: una cancellazione accidentale, una goccia di caffè o anche una caduta di tensione mentre state leggendo o scrivendo possono danneggiare tanto il dischetto quanto i file in esso contenuti.
Per prevenire le conseguenze di questa imprevedibile catastrofe, potete fare una copia di questo dischetto e poi usare questa copia al posto dell'originale. Se qualcosa dovesse succedere alla copia, potete farne un'altra dall'originale. I comandi FORMAT e COPY vi permettono di fare copie del disco di sistema dell'MSX-DOS.

Una volta formattato un nuovo disco, il passo successivo consiste nel copiare tutti i file dal disco di sistema al nuovo.

Per esempio, se avete un computer con due drive, mettete prima il dischetto di sistema nel drive A: e un disco nuovo e pulito nel drive B:. Poi date il comando

```
A>FORMAT
```

Il dischetto ora non contiene ancora nessuno dei file dell'MSX-DOS mostrati nell'elenco del directory. Per copiare questi file, date il comando

```
A>COPY A:*.* B:
```

Se avete un solo drive, dovete sostituire i dischetti durante la copia di ogni file.
Ora potete usare il dischetto nel drive B: per far partire il computer.
Gli asterischi usati nel comando COPY sono descritti più avanti e servono ad indicare all'MSX-DOS di copiare tutti i file presenti in A:, a prescindere dai loro nomi.

PROTEZIONE DEL DISCO DI SISTEMA

Ora che avete una copia di riserva del disco di sistema dell'MSX-DOS, potete proteggerla da cancellazioni accidentali o da modifiche non desiderate, mediante un sistema di protezione contro la sovrascrittura.
Nei dischetti da 3"1/2, il forellino nell'angolo in alto a destra si usa come protezione dalla sovrascrittura del dischetto. Il metodo di protezione per questi dischi dipende dalla marca dei dischi stessi. Alcuni hanno uno sportellino mobile, mentre altri hanno una linguetta che si deve spezzare per scoprire il forellino.
È buona regola proteggere i dischetti su cui non volete scrivere: sia i dischi originali che le copie del dischetto di sistema. Naturalmente, potete togliere la protezione e scrivere informazioni sul disco, se necessario. Lo scopo fondamentale della protezione di un dischetto è di ridurre le possibilità di distruggere accidentalmente delle informazioni importanti.

18.5 Uso dei file nei comandi e nei programmi

I comandi MSX-DOS vi serviranno soprattutto per la gestione dei file. Naturalmente c'è ancora molto da imparare sui comandi, ma per usare l'MSX-DOS efficacemente sono sufficienti pochi comandi, che presenteremo brevemente qui di seguito.

CREARE FILE

I file vengono sempre creati da programmi. Un programma può contenere le istruzioni per la creazione di un file (per esempio un testo creato da un programma di word processing) oppure può creare file che userà esso stesso. Qui illustriamo il metodo generale per creare i file senza addentrarci nella descrizione dei comandi particolari.
Un modo per creare file è di usare un word processor. Un altro è di creare un file con l'MSX-BASIC. La maggior parte dei programmi applicativi crea dei file; per esempio, un programma di contabilità crea un file che contiene i conti e un foglio elettronico produce un file che contiene i modelli finanziari. Inoltre, se usate un linguaggio di programmazione diverso dall'MSX-BASIC, vi sarà possibile salvare i programmi come file su disco.

COME DARE I NOMI AI FILE

Per memorizzare un file su disco, dovete dargli un nome cui si possa fare riferimento in modo univoco e che è perciò chiamato specificatore del file. Lo specificatore del file è diviso in due parti: il *nome del file* e la sua *estensione*. Ogni file deve avere un nome ma non necessariamente un'estensione. Il nome e l'estensione servono per descrivere il contenuto del file. La combinazione di queste due parti non si deve ripetere due volte nello stesso disco, cioè non possono esistere sullo stesso disco due file con lo stesso specificatore. La regola da ricordare è che il nome del file può essere al massimo di otto caratteri e l'estensione di tre.
Il nome del file e l'estensione sono sempre separati da un punto. Per esempio, uno specificatore di file può essere NUMTEL.DAT: NUMTEL è il nome del file e DAT è l'estensione. Altri esempi di specificatori di file sono: CONTAB.BAS, LETTERA3.TXT o PROFITTI (in questo caso l'estensione è stata omessa). Quando chiedete all'MSX-DOS di leggere o scrivere un file, dovete fornirgli il nome del file e, se esiste, la sua estensione.
Fate attenzione, perché in alcuni manuali il termine "nome del file" è sinonimo di "specificatore di file" e quindi include anche l'estensione. Quest'ultima viene spesso chiamata anche "tipo di file".

Caratteri legali

Lettere (solo maiuscole):

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Cifre:

0 1 2 3 4 5 6 7 8 9

Caratteri speciali e segni di punteggiatura:

! @ # $ % ^ & ( ) { } - _ ' ~

Caratteri illegali

, . / \ | ? * " : ; [ ] + =

Figura 18.1 Caratteri legali e non per specificatori di file

Nel dare i nomi ai file avete un'ampia libertà, ma tenete a mente le seguenti regole quando scegliete nomi ed estensioni.

- Non tutti i caratteri che vedete sulla tastiera possono essere usati per formare i nomi dei file, perché alcuni di essi sono impiegati dall'MSX-DOS e potrebbero creare confusione. I caratteri che potete e che non potete usare sono mostrati nella Figura 18.1.
- Scegliete un nome per il file che richiami il più possibile il contenuto. Per esempio, una lettera a una persona che si chiama Rossi può essere chiamata ROSSILET.TXT; l'elenco dei vostri fornitori può chiamarsi FORNIT.DAT. In questo modo è più facile risalire immediatamente al contenuto del file senza doverlo richiamare sullo schermo.
- L'MSX-DOS non vi permette di dare ai file nomi uguali a quelli delle periferiche (per esempio la stampante). Perciò non potete usare i nomi AUX, CON, LST, PRN NUL.
- Mettete sempre un'estensione nello specificatore di file, anche se non è richiesta: sarà un'ulteriore informazione sul contenuto del file che potrà risultarvi utile in un secondo tempo. Per esempio, se dovete scrivere una lettera alla Edsel Corporation invece di chiamare il file EDSEL, potete chiamarlo EDSEL.LET o EDSEL.TXT. Se poi create un altro file con la contabilità della Edsel, potete chiamarlo EDSEL.DAT. L'estensione vi permette di distinguere facilmente i due file EDSEL.
- Date nomi simili a file che riguardano lo stesso oggetto per facilitare la ricerca su disco di un file o di un gruppo di file. Potete così chiamare le vostre lettere al signor Bianchi: BIANCHI1.LET, BIANCHI2.LET e così via.

Estensione	**Significato**
ASM	Programma sorgente in Assembler
BAK	Copia di riserva
BAS	Programma BASIC
BAT	File batch
BIN	File binario
C	Programma sorgente in C
COB	Programma sorgente in COBOL
COM	Programma
DAT	File di dati
EXE	Programma
OBJ	File oggetto
OVR	File supplementare per programma applicativo
TEX	File di testo
TXT	File di testo

Figura 18.2 Estensioni usuali di file

Specificatori validi

MODEM.DOC
CAP-3A.TXT
TELEF.DAT
10IDEA
XX!!!@@@ (non consigliabile)

Specificatori non validi

COMPUTERS.DOC	(Nome del file più lungo di 8 caratteri)
CAP-3A.TEXT	(Estensione più lunga di 3 caratteri)
TELEF.NOMI.DAT	(Due estensioni)
10IDEA?	(Carattere non ammesso)

Figura 18.3 Esempi di specificatori validi e non validi

- Dove possibile, usate le estensioni standard. La Figura 18.2 elenca alcune estensioni convenzionali usate sui microcomputer. Naturalmente potete usare i nomi che volete ignorando queste convenzioni, ma alcuni programmi presuppongono che i file abbiano certe estensioni e potrebbero creare difficoltà.
- Evitate di usare estensioni che abbiano un significato speciale (come COM, EXE o BAT), perché l'MSX-DOS potrebbe scambiare il file per un comando e cercare di eseguirlo con risultati disastrosi. (Per inciso, potete vedere quali file su un dato disco sono comandi, programmi o file batch guardando la loro estensione: tutti i comandi hanno l'estensione COM o EXE, mentre i file batch hanno l'estensione BAT.)
- Anche se è lecito usare molti segni di interpunzione, in genere è bene limitare il loro impiego. Il simbolo del dollaro ($), il trattino (–) e la sottolineatura (_) dovrebbero bastarvi per definire un nome. In Figura 18.3 sono elencati alcuni specificatori di file validi e non validi.

USO DEI CARATTERI JOLLY PER RAGGRUPPARE I FILE

Potete usare alcuni comandi MSX-DOS per lavorare su un gruppo di file con lo stesso nome o con la stessa estensione. Il raggruppamento dei file vi permette di risparmiare tempo e istruzioni. Usare un nome comune per i file è come usare il cognome per una famiglia; invece di dire: "Voglio incontrare Roberto Rossi, Giovanna Rossi, Diana Rossi e Renzo Rossi", potete dire: "Voglio incontrare la famiglia Rossi".
Così se voleste trasferire le copie di tutti i vostri 40 programmi in BASIC da A: a B: dovreste dare il comando 40 volte successivamente:

```
A>COPY PROG1.BAS B:

A>COPY PROG2.BAS B:
```

e così via.
Per risparmiarvi questo enorme lavoro, l'MSX-DOS vi permette di usare due caratteri speciali che si utilizzano per specificare più di un file: il punto di domanda (?) e l'asterisco (*). Questi si usano negli specificatori di file per far agire l'MSX-DOS su un gruppo di file anziché su un singolo file. Invece di riprodurre i nomi dei file lettera per lettera, l'uso di questi caratteri fa sì che l'MSX-DOS ricerchi file che hanno dei caratteri qualsiasi nella parte del loro specificatore occupata da ? o da *. Come un jolly nel gioco delle carte, questi due caratteri possono essere usati per rappresentare un qualsiasi carattere della tastiera e li chiameremo perciò caratteri jolly.
Il punto di domanda viene usato al posto di un carattere qualsiasi che si trovi nella posizione in cui esso compare all'interno del nome del file.

Per esempio, se volete copiare tutti i file il cui nome è lungo cinque lettere, di cui le prime quattro sono PROG, e che hanno l'estensione BAS, potete dare il comando

```
A>COPY PROG?.BAS B:
```

Il comando copia PROG1.BAS, PROG2.BAS, PROGA.BAS, PROGB.BAS e così via, dal drive A: sul dischetto nel drive B:. Non copia invece né PROG10.BAS né PROGRAM.BAS poiché i loro nomi hanno più lettere di PROG?.BAS e il punto di domanda rappresenta un solo carattere. Un altro esempio è

```
A>COPY TAX?83.DAT B:
```

Questo comando copia i seguenti file: TAXA83.DAT, TAX183.DAT, TAX-83.DAT, ecc., ma non copia TAXAB.DAT o TAXA.DAT.

L'asterisco, invece, sostituisce un numero qualsiasi di caratteri che si trovano nella posizione in cui esso compare nel nome. Ciò significa che il numero di lettere che state sostituendo è irrilevante e che potete selezionare file con lettere diverse nella posizione che indicate col carattere *.

Per esempio, per copiare un file qualsiasi il cui nome inizi con le lettere PROG e la cui estensione sia BAS, date il comando

```
A>COPY PROG*.BAS B:
```

Vengono copiati file del tipo PROG1.BAS, PROG1A.BAS, PROG9999.BAS e PROG2B2B.BAS.

Come potete vedere confrontando i due ultimi esempi, usando l'asterisco ci si riferisce almeno a tutti i file cui si fa riferimento con il punto di domanda.
In uno specificatore di file è possibile usare più di un carattere jolly. Per esempio, B???E.* si riferisce a file del tipo BOLLE.TXT, B000E.123, B-D-E.TXT e BRAKE.
L'uso dei caratteri jolly semplifica il problema di copiare i programmi da A: a B:. Infatti per copiare tutti i file con estensione BAS, a prescindere dalla lunghezza del loro nome, potete rimpiazzare il nome con un asterisco dopo il comando COPY:

```
A>COPY *.BAS B:
```

Se, invece, volete copiare tutti i file in B:, senza tener conto del nome o dell'estensione, dovete dare il comando

```
A>COPY *.* B:
```

Questa è la stessa procedura già usata per copiare il dischetto di sistema.

USO DEI NOMI DI PERIFERICHE NEI COMANDI

Come avete visto, potete usare il comando COPY per copiare file su o da disco. Potete anche usarlo per spostare informazioni su o da una periferica. L'MSX-DOS vi fornisce un modo estremamente facile per comunicare con la stampante, le porte di comunicazione e le altre periferiche.
È molto improbabile che dobbiate usare nomi di periferiche col comando COPY perché la maggior parte dei programmi che utilizzano periferiche, comunica con queste ultime automaticamente attraverso comandi interni. I nomi sono elencati nella Figura 18.4. Tutti i nomi delle periferiche sono lunghi tre caratteri.
Invece di usare il nome di un file dopo il comando COPY, potete usare il nome di una periferica; infatti proprio come un file, ogni periferica può essere usata come input (per leggere) o come output (per scrivere). Di fatto la comunicazione con le periferiche è simile all'azione di copiatura di dati verso e da un file e quindi non dovrebbe presentare particolari difficoltà.
L'unico comando con cui si usano nomi di periferiche è il comando COPY.

Nome	**Significato**
CON	Console, cioè la combinazione di tastiera e schermo
AUX	Primo porta seriale di comunicazione o porta RS232
LST o PRN	Porta di stampa parallela *Centronics*
NUL	Periferica inesistente, usata solo per provare i programmi

Figura 18.4 Nomi di periferiche

Per esempio, se volete inviare un file alla stampante, date il comando

```
A>COPY FILEMIO.TXT PRN:
```

Questo provoca la stampa del file FILEMIO.TXT che si trova nel drive A:, sulla stampante.

18.6 Uso dei file batch per associare più comandi

Ora che conoscete un certo numero di comandi MSX-DOS e sapete in generale come usarli, siete pronti per imparare ad eseguire un blocco che contenga più comandi uno dopo l'altro. Se dovete eseguire molti comandi, potete risparmiare molto tempo riunendoli in un file di testo chiamato *file batch*. Quando eseguite questo file vengono eseguiti tutti i comandi che esso contiene proprio come se li batteste in quel momento.
Scrivere un file batch è come scrivere un programma molto semplice che fornisce all'MSX-DOS una lista di comandi da eseguire. È comunque molto più semplice che programmare. Quando volete che i comandi vengano eseguiti, date all'MSX-DOS il nome del file batch invece dei nomi di tutti i comandi.
In un file batch potete usare qualsiasi programma o comando che usereste normalmente dopo il messaggio A>. Vedrete che combinare i comandi in questo modo, vi farà risparmiare moltissimo tempo di battitura.

Per esempio, supponete di usare un programma di gestione detto CONTI, che crea un file di nome VENDITE.DAT. Dovete far eseguire il programma ogni giorno e, ad ogni esecuzione di CONTI, eseguire anche il comando COPY per trasferire il contenuto del file VENDITE.DAT nel file NUOVEVEN.DAT. Infine il programma CONTI deve essere seguito anche dai nomi degli altri due file con cui lavora e cioè COGE.DAT e NUOVAREL.TXT. Se non usate un file batch dovete dare i comandi

```
A>CONTI VENDITE.DAT COGE.DAT NUOVAREL.TXT

A>COPY VENDITE.DAT NUOVEVEN.DAT

A>
```

Potete però ridurre il lavoro di battitura creando un file batch il cui nome, per esempio, sarà GIORNO.BAT; per far questo si deve usare COPY con *nomefile* (cioè si crea il file da console) o un editor:

GIORNO.BAT

```
CONTI VENDITE.DAT COGE.DAT NUOVAREL.TXT
COPY VENDITE.DAT NUOVEVEN.DAT
```

Ora, quando volete far eseguire il vostro file batch, è sufficiente che diate all'MSX-DOS il semplice comando GIORNO (non è necessario scrivere anche l'estensione BAT). Quando l'MSX-DOS trova il file batch chiamato GIORNO.BAT inizia ad eseguire i comandi:

```
A>GIORNO

A>CONTI VENDITE.DAT COGE.DAT NUOVAREL.TXT

A>COPY VENDITE.DAT NUOVEVEN.DAT

A>
```

Notate che l'MSX-DOS prima ha trovato il file batch e poi ha eseguito tutte le linee che lo costituiscono, mostrandole via via sullo schermo.

REGOLE PER DARE I NOMI AI FILE BATCH

Ci sono alcune regole da seguire per dare un nome a un file batch:

- Al nome di un file batch deve essere aggiunta l'estensione BAT, in modo che il file possa essere riconosciuto dall'MSX-DOS come file batch.
- Non date ad un file batch lo stesso nome di un comando. Se lo faceste, l'MSX-DOS manderebbe sempre prima in esecuzione il comando.
- Ai nomi dei file batch si applicano le stesse regole sui caratteri che valgono per tutti gli altri file.

Potete mettere in un file batch un comando qualsiasi e il file può avere

una lunghezza qualunque. Ogni comando deve occupare una linea separata, perché l'MSX-DOS legge i comandi dal file batch proprio come se li aveste battuti dopo il messaggio A>.
Ci sono poi comandi speciali che si usano solo con i file batch per rendere più semplice la loro manipolazione. Tali comandi, che permettono di sostituire i dischi, sapere se esiste un certo file e così via, sono descritti nel paragrafo 19.9 "Comandi per file batch".

USO DI ARGOMENTI ALL'INTERNO DI FILE BATCH

Gli esempi precedenti riguardano l'uso di file batch al posto di un insieme di comandi eseguiti sempre con gli stessi argomenti. Cosa fare se volete fornire ad un comando in un file batch un argomento sempre diverso? L'MSX-DOS vi permette di realizzare a questo scopo una sostituzione di argomenti. Potrete così fornire nuovi argomenti ai comandi che si trovano nel file batch. Casualmente per l'MSX-DOS non ha importanza il tipo di argomento fornito: può trattarsi del nome di un file, di un drive, o di un qualsiasi altro argomento.
Per sostituire un argomento in una riga corrispondente a un comando di un file batch, dovete usare il carattere % e il numero dell'argomento nel file (per esempio, al primo argomento ci si riferisce come %1). Il simbolo di percentuale indica all'MSX-DOS di rimpiazzare %1 con il primo argomento che battete quando fate eseguire il file batch; invece %2 viene rimpiazzato con il secondo argomento battuto e così via. Nel seguente esempio è illustrato un caso in cui è utile sostituire gli argomenti.

Supponete di dover copiare un file sul disco nel drive A: con un nome e sul disco nel drive B: con un altro. Potete creare un file batch che svolge entrambe le azioni. Questo sarà:

Al momento di mandare in esecuzione il file batch RPTCOPY.BAT fornite i due nomi coi quali volete memorizzare il file sui dischi. Ad esempio:

```
A>RPTCOPY.BAT TESTO1 TESTO2

A>COPY TESTO.TXT A:TESTO1.TXT
        1 file copied
A>COPY TESTO.TXT B:TESTO2.TXT
        1 file copied
A>
```

Se avete molti comandi che usano lo stesso argomento ogni volta che eseguite il vostro file batch, potete usare lo stesso simbolo di percentuale più di una volta.

Per esempio, supponiamo che eseguiate ogni volta il comando CALC-INI insieme al comando CALC-FIN. Entrambi i comandi prendono lo stesso argomento e cioè il nome di un file di dati. Potete scrivere un file batch di nome CALC.BAT per eseguirli usando lo stesso argomento.

CALC.BAT

```
CALC-INI %1
CALC-FIN %1
```

Quando mandate in esecuzione questo file con argomento FY1982.DAT vedete il seguente risultato:

```
A>CALC FY1982.DAT

A>CALC-INI FY1982.DAT

A>CALC-FIN FY1982.DAT

A>
```

Se volete in un comando più argomenti potete usare più simboli di percentuale: %2, %3, %4 e così via. Infatti l'MSX-DOS vi permette di assegnare fino a nove argomenti ad un file batch. Proprio come prima, se usate un secondo argomento, l'MSX-DOS lo utilizzerà per rimpiazzare ogni %2 nel vostro file batch, rimpiazzerà ogni %3 con il terzo argomento e così via.
La possibilità dell'MSX-DOS di includere molti argomenti nei file batch è un importante strumento con cui potete creare file che eseguono compiti complessi.

Supponete di aver bisogno di fornire un nome al secondo e al terzo file nel file batch GIORNO.BAT usato in un esempio precedente. Dovreste riscrivere il file così che includa due argomenti da sostituire:

Date ora il comando GIORNO e i due nomi dei file e il risultato sarà:

```
A>GIORNO RICEVI.DAT RCVREL.TXT

A>CONTI VENDITE.DAT RICEVI.DAT RCVREL.TXT

A>COPY VENDITE.DAT NUOVEVEN.DAT

A>
```

Ora che la sostituzione di argomenti nei file batch vi è familiare, potete rendervi conto del perché dovete evitare di usare il simbolo di percentuale nei nomi dei file. Se nonostante tutto aveste uno specificatore di file nel vostro programma batch che contiene tale simbolo (per esempio N%X.DAT), dovete ricordarvi di aggiungere un altro simbolo identico (N%%X.DAT): in questo modo l'MSX-DOS saprà che il primo simbolo di percentuale va inteso come parte di un nome di file.

IL FILE AUTOEXEC.BAT

Rimane da discutere ancora un tipo di file batch e precisamente quel file batch che viene eseguito automaticamente (si "autoesegue") ogni volta che si accende il computer: si chiama appunto AUTOEXEC.BAT. Questo file può essere scritto al fine di eseguire automaticamente programmi, oppure per stampare i risultati ottenuti da programmi eseguiti il giorno prima non appena accendete il computer.
Un file AUTOEXEC.BAT viene creato allo stesso modo di un file batch; gli viene dato però il nome di AUTOEXEC.BAT. Potete inserirgli qualsiasi comando vogliate. Ogni volta che caricate l'MSX-DOS esso ricerca il file AUTOEXEC.BAT e, dopo averlo trovato, esegue automaticamente i comandi. Per inciso va detto che questo è il file speciale di cui abbiamo parlato all'inizio del capitolo, quel file, cioè, che impedisce all'MSX-DOS di chiedervi la data.
Nel file AUTOEXEC.BAT vengono posti in genere i comandi che mettono a punto le inizializzazioni del sistema (come la data e l'ora); potete però anche aggiungervi programmi che devono essere caricati in memoria immediatamente dopo l'accensione del sistema.

18.7 Esempio di sessione MSX-DOS

A questo punto conoscete tutto ciò che è necessario per incominciare ad usare l'MSX-DOS. Presenteremo perciò un tipico esempio di sessione al computer utilizzando l'MSX-DOS.
Supponiamo che nel corso della giornata dobbiate usare un programma di controllo dell'inventario, e che dobbiate preparare un rapporto sui risultati. Dovete inoltre stampare tale rapporto e copiarlo su un dischetto insieme con la relazione. Tutte queste funzioni sono qui di seguito mostrate, passo dopo passo.

FASE 1: USO DI UN FILE BATCH PER FAR ESEGUIRE IL PROGRAMMA

Prima di tutto vogliamo eseguire il programma di inventario. Per facilitare questa azione avete già costruito un file batch (INVREL) che usa tre diversi comandi. Esso contiene

INVREL.BAT

```
INVMAKE %1 %2 INVKEY.TXT
INVFILE INVKEY.TXT
INVOUT %1 %2 INVKEY.TXT
```

Tutto ciò che dovete fare è dire al file batch quali sono le date di inizio e di fine relazione.

```
A>INVREL 1/8/84 15/8/84

A>INVMAKE 1/8/84 15/8/84 INVKEY.TXT

A>INVFILE INVKEY.TXT

A>INVOUT 1/8/84 15/8/84 INVKEY.TXT
Il rapporto e'salvato nel file INVRPT93.TXT

A>
```

L'ultimo messaggio ("Il rapporto è salvato...") è stato stampato dal programma INVOUT che, nel nostro esempio, rappresenta un programma che indica, con tale messaggio, il nome del file che ha prodotto e salvato su disco (in questo caso INVRPT93.TXT). Quando il programma è finito ricompare A>.

FASE 2: ANALISI DEI RISULTATI

Il passo successivo consiste nell'analisi dei risultati appena prodotti. Potete farlo sullo schermo o stampare il file. Per visualizzare i risultati sullo schermo usate il comando TYPE.

```
A>TYPE INVRPT93.TXT

            Rapporto di inventario dal 1/8/84 al 15/8/84
```

```
Scorte da ordinare per il 20/8/84:

Articolo     Giacenza     Scorta minima     Prezzo
- - - - - - - - - - - - - - - - - - - - - - - - - -
CF3499          8              12            25750
CF5000         52              60             2200
   .            .               .               .
   .            .               .               .
   .            .               .               .
```

Se il rapporto risulta troppo lungo, potete stamparlo con il comando COPY e il nome di periferica PRN

FASE 3: SCRITTURA DELLA RELAZIONE

La relazione che espone in dettaglio i risultati dell'analisi appena compiuta va scritta attraverso un word processor. Ogni word processor usa comandi diversi e quindi non spiegheremo qui ogni fase della scrittura. Supponete di utilizzare un word processor detto TEXTP e di voler lavorare con il file INVRPT93.TXT. Dovete fornire il comando

```
A>TEXTP INVRPT93.TXT
```

Aggiungete un certo testo alla relazione prodotta e riordinate i risultati. Dopo di che richiedete una stampa della relazione su stampante. Non tutti i word processor, però, consentono la stampa del testo riordinato.

FASE 4: COPIATURA DEI FILE CONTENENTI LA RELAZIONE

Si vuole ora completare il lavoro copiando su disco la relazione. Bisognerà inserire un disco formattato sul secondo drive (B:) e assicurarsi che sia vuoto con il comando DIR.

```
A>DIR B:

File not found

A>
```

Ora date i comandi di copiatura dei due file:

```
A>COPY INVRPT93.TXT B:
        1 file copied
```

```
A>COPY INVNUOV.MEM B:
          1 file copied

A>
```

Il file INVNUOV.MEM è il file scritto attraverso il word processor. Ricordate che il comando COPY vuole due argomenti: la sorgente e la destinazione dei dati. Nei casi indicati la sorgente è un file in A: (il drive di default) e la destinazione è il drive B:. Per assicurarvi di aver copiato tutto correttamente potete analizzare il contenuto di B: con DIR.

```
A>DIR B:

Insert diskette for drive B:
and strike a key when ready
INVRPT93 TXT     8110 1-01-84
INVNUOV  MEM      631 1-01-84
       2 files    151515 bytes free
A>
```

Il comando vi dà, per ogni file, il suo nome, la sua estensione, il numero di byte che contiene e la data. Vi dice anche quanti file sono sul disco e lo spazio ancora disponibile ("bytes free"). Il vostro compito è finito. Dovreste sentirvi ormai a vostro agio nell'usare i più semplici comandi MSX-DOS.

Capitolo

Comandi MSX-DOS 19

19.1 Organizzazione del capitolo

Questo capitolo descrive i comandi MSX-DOS e vi suggerisce come usarli utilizzando numerosi esempi.
I comandi vengono presentati riuniti in gruppi rispetto alla loro funzione. Per esempio, tutti i comandi usati per la gestione dei file (come la loro copiatura o cancellazione) sono raggruppati insieme.
Ogni descrizione di un comando comprende le seguenti informazioni:

- **Usi comuni.** Viene mostrato come usare in diversi modi i comandi e tutti i loro argomenti. La maggior parte degli esempi si trova in questa sezione. Alcuni comandi particolarmente complessi contengono anche una sezione chiamata "Altri usi" che descrive usi meno comuni del comando.
- **Regole.** Vengono spiegate le regole che governano l'uso di alcuni comandi.
- **Avvertimenti ed errori comuni.** Questa parte elenca problemi che sono comuni ai comandi specifici e fornisce suggerimenti che possono aiutarvi ad evitare grossi problemi.
- **Sintassi.** La sintassi indica la combinazione ordinata degli argomenti di un comando. Questa sezione, che elenca la sintassi completa per ogni comando, contiene di solito molte più informazioni di quelle che vi sono necessarie normalmente, ma che potranno esservi utili per capire esattamente l'uso del comando. Sintassi di tipo generale sono fornite nella parte "Usi comuni".

19.2 Come leggere la sintassi di un comando

L'uso corretto dei comandi richiede la conoscenza del linguaggio MSX-DOS. Il sistema è in grado di leggere i comandi solo se il loro nome e i loro argomenti si presentano nella combinazione corretta, specificata nella loro sintassi. La sintassi può essere corta e semplice oppure lunga e complessa, e specifica ogni cosa di cui il comando ha bisogno per essere eseguito. Se leggete attentamente la sintassi e fornite tutte le informazioni attraverso gli argomenti, potete star certi che il comando non darà problemi.
I tre concetti che seguono sono importanti per capire come interpretare le notazioni riguardanti la sintassi, usate in questo libro. Notate che la sintassi dei comandi MSX-DOS è differente da quella dell'MSX-BASIC precedentemente vista.

- **Simboli speciali.** Indicano tipi diversi di argomenti e sono elencati nella Figura 19.1. Ogni argomento che non è compreso nei simboli speciali è richiesto espressamente per il comando in questione. Potete vedere che la maggior parte degli argomenti è opzionale.
- **Argomenti in corsivo.** In corsivo sono indicati gli argomenti generici, cioè quelli che possono assumere molti valori. Per esempio, nella sintassi TYPE *specfile*, *specfile* indica che in quel punto può essere usato ogni nome o specificatore di file. L'argomento *specfile* segnala qual è il posto destinato al nome di file.
- **Argomenti di una sola lettera.** Sono chiamati anche "opzioni"; la lettera è sempre preceduta da una barra (per esempio /V).

Gli esempi seguenti mostreranno quanto sia importante comprendere la sintassi dei comandi.

Il primo esempio è

DIR [*d:*]

Con questa sintassi DIR può essere lasciato senza argomenti oppure essere seguito da un identificatore di drive come B:. Un altro esempio è

TYPE *specfile*

Questo comando richiede uno *specfile*, cioè il nome di un file. Ricordate che *specfile* può includere anche l'identificatore di drive.

Simbolo	Significato
[ ]	Parentesi quadre: indicano che tutto ciò che è al loro interno è opzionale
{ }	Parentesi graffe: indicano che dovete scegliere solo una delle alternative contenute al loro interno. Ogni alternativa è separata da una barra verticale (|)
...	Puntini: indicano che potete ripetere l'ultimo insieme di argomenti

Figura 19.1 Simboli usati nella sintassi dei comandi

19.3 Come fermare un comando

In alcuni casi può rendersi necessario fermare un comando in esecuzione, per esempio se il comando sta stampando informazioni che non vi interessano. Potete far questo tenendo premuto il tasto CTRL e premendo contemporaneamente il tasto STOP; sullo schermo dovrebbe comparire il solito prompt A>. Ci sono alcuni casi sfortunati in cui questo metodo non funziona.

Se dovete fermare un comando e il metodo appena descritto non funziona, come ultima risorsa potete sempre spegnere il computer. Spegnendo il computer obbligate la macchina ad interrompere quello che sta facendo senza badare alle conseguenze di tale azione.

Spegnere il computer per bloccare un comando non è però sempre raccomandabile: infatti, se il computer sta scrivendo su disco, è possibile danneggiare in modo serio le informazioni ivi contenute.

Se si desidera fermare un comando solo temporaneamente, senza far comparire il prompt A>, ad esempio se il comando sta mostrando sullo schermo un gran numero di informazioni, è possibile fermare il processo premendo il tasto STOP. Premendo il tasto STOP una seconda volta potete far riprendere l'esecuzione del comando.

19.4 Come passare in MSX-BASIC

Per avviare l'MSX-BASIC quando si è in ambiente DOS è sufficiente dare il semplice comando

BASIC

Questo farà comparire la scritta di presentazione dell'MSX-BASIC ed il suo prompt Ok.
Per l'operazione inversa, cioè per passare dall'MSX-BASIC al sistema operativo date il comando

CALL SYSTEM

o

_SYSTEM

Ricordate che tutto ciò che stavate elaborando in BASIC andrà perduto al momento del passaggio all'MSX-DOS.

19.5 Comandi per la gestione dei file

Questa sezione illustra i comandi ERASE, DEL, RENAME, REN e COPY.

ERASE e DEL

Il comando ERASE cancella un file o un gruppo di file dal directory di un disco ed è identico al comando DEL.

Usi comuni

Ci sono diverse situazioni in cui usare il comando ERASE: ad esempio, se uno dei vostri programmi ha creato un file inutile, oppure se avete su disco un file che non vi serve più. Poiché lo spazio in memoria è limitato, è importante cancellare i vecchi file per poterne inserire dei nuovi.
La sintassi generale per il comando ERASE è

ERASE *specfile*

Il processo di cancellazione di un file è semplice e richiede solo il nome del file da cancellare.

Per esempio per cancellare un file di nome BOSTON88.TST sul drive di default date il comando

```
A>ERASE BOSTON88.TST
```

oppure

```
A>DEL BOSTON88.TST
```

e il file verrà rimosso dal disco e dal directory.

Potete usare caratteri jolly per cancellare più di un file alla volta.

Per cancellare tutti i file con estensione BAK (usata per i file di backup) date il comando

```
A>ERASE *.BAK
```

Per cancellare tutti i file su un disco, il comando ERASE vi chiederà prima una conferma.

```
A>ERASE *.*
Are you sure (Y/N)?
```

Il comando vi chiede infatti se siete sicuri di ciò che chiedete e voi dovete rispondere "sì" (Y) o "no" (N).
Verificate prima attentamente il contenuto dei file da cancellare, perché dopo l'esecuzione del comando non c'è più la possibilità di recuperarlo.

Regole

Dovete sempre specificare quali file volete cancellare. Se dimenticaté di farlo l'MSX-DOS vi darà un messaggio d'errore.

Avvertimenti ed errori comuni

Dovete fare attenzione, naturalmente, a non cancellare file importanti. Se condividete l'uso del computer con altre persone state attenti a cancella-

re un qualsiasi file, perché qualcun altro può aver bisogno di un file che voi ritenete poco importante o inutile.

Sintassi

La sintassi del comando ERASE è

ERASE *specfile*

Con il comando deve essere indicato sempre *specfile* che può contenere caratteri jolly.

RENAME e REN

Il comando RENAME cambia il nome di un file senza modificarne il contenuto. È identico al comando REN.

Usi comuni

Ci sono varie occasioni in cui può essere utile usare questo comando: il contenuto di un file può essere cambiato talmente che il vecchio nome non risulta più adatto, oppure potete accorgervi di aver dato incidentalmente a due file nomi simili e volete cambiare il nome di uno di essi per evitare confusioni. La sintassi corrente è

RENAME *specfile1 specfile2*

specfile1 sta per il nome originale del file mentre *specfile2* è il nome nuovo che gli volete dare.

Per esempio, per dare il nome VENDEXP.DAT al file VENDITE.DAT date il comando

```
A>RENAME VENDITE.DAT VENDEXP.DAT
```

oppure

```
A>REN VENDITE.DAT VENDEXP.DAT
```

Potete usare questo comando insieme a ERASE per eliminare la versione corrente di un file e rimpiazzarlo con una vecchia nel caso vi siate accorti, rivedendo il file, che quest'ultima è più appropriata.

Supponiamo che vogliate dare un nuovo nome al programma backup di un file di fatture detto FATTPROG.COB. State usando un editor che salva le copie dei file con lo stesso nome degli originali, ma con estensione BAK. Darete allora i comandi

```
A>ERASE FATTPROG.COB

A>RENAME FATTPROG.BAK FATTPROG.COB
```

Se voleste semplicemente scambiare i nomi dei file dovreste usare un nome temporaneo perché il comando non vi permette di cambiare contemporaneamente i nomi di due file. Potreste aver bisogno di cambiare i nomi dei file se aveste deciso di usare la versione originale di un file anziché quella rivista, pur mantenendo quest'ultima per un uso futuro.

Per esempio per cambiare i nomi dei file VENDPROG.COB e VENDPROG.BAK battete i seguenti comandi

```
A>RENAME VENDPROG.COB TEMPFILE.COB

A>RENAME VENDPROG.BAK VENDPROG.COB

A>RENAME TEMPFILE.COB VENDPROG.BAK
```

TEMPFILE.COB è il nome di un file temporaneo.

Se avete su un disco due o più file con lo stesso nome o estensione, potete usare caratteri jolly per dare nomi nuovi a questi file nel loro complesso.

Con questo primo esempio si dà un nome nuovo COGE a tutti i file di nome CONTABIL.

```
A>RENAME CONTABIL.* COGE.*
```

Questo secondo esempio, invece, sostituisce l'estensione TEX a tutti i file con estensione TXT.

```
A>RENAME *.TXT *.TEX
```

Regole

Non potete usare RENAME per cambiare i nomi di un gruppo di file distribuiti su dischi diversi: dovete usare il comando COPY per concentrarli tutti sullo stesso disco.

Non potete dare a un file un nome che appartiene ad un altro file sullo stesso disco. Se lo fate l'MSX-DOS risponde con il messaggio "Rename error".

Avvertimenti ed errori comuni

L'errore più comune è quello di confondere l'ordine dei due nomi di file: RENAME richiede che prima venga il nome vecchio e poi quello nuovo. Se molte persone usano il vostro computer, assicuratevi che quando date un nuovo nome a un comando o a un file, ognuna di queste persone venga a conoscenza del nome nuovo per non causare inutili confusioni.
Non cambiate i nomi di file speciali usati dai programmi applicativi, altrimenti i programmi stessi non sarebbero in grado di rintracciare i file di cui hanno bisogno. Il programma WordStar, per esempio, richiede la presenza sul disco del file WSOVLY1.OVR.

Sintassi

Le seguenti sintassi del comando RENAME sono entrambe corrette:

RENAME *specfile1 specfile2*

oppure

RENAME *specfile1 nomefile2*

Nella prima forma, *specfile1* è il nome del file cui state dando un nuovo nome e *specfile2* è il nuovo nome. Nella seconda forma *nomefile2* è il nuovo nome del file e, mancando l'estensione, l'MSX-DOS userà la stessa estensione del file originale. Notate che il vecchio nome viene sempre dato per primo.

COPY

Il comando COPY vi permette di copiare file da un disco ad un altro, di realizzare la copia di un file sullo stesso disco (ma con un nome diverso), di scambiare informazioni con le periferiche di sistema e di riunire insieme più file.

Usi comuni

COPY è uno dei comandi più usati nell'MSX-DOS, soprattutto per copiare file da un disco ad un altro.
La sintassi generale è

COPY *sorgente destinazione*

dove *sorgente* è il nome del file che volete copiare e *destinazione* è il luogo dove volete copiarlo.

Se volete, per esempio, copiare il file CRED.DAT da un disco sul drive A: in un disco sul drive B: dovete dare il comando

```
A>COPY CRED.DAT B:
           1 file copied
```

In questo modo il file viene copiato con lo stesso nome sul drive B:.

Come avete potuto notare il comando COPY vi indica sempre quanti file ha copiato. Inoltre non è necessario dare il nome del file sul disco di destinazione ma solo il nome del drive. In tal caso il comando presume che vogliate dare al file copiato lo stesso nome che aveva l'originale.
Per cambiare il nome al file copiato basta aggiungere semplicemente al comando il nuovo nome.

Per esempio:

```
A>COPY CRED.DAT B:NVCRED.DAT
           1 file copied
```

In questo caso non solo copiate CRED.DAT da A: in B:, ma date anche al file sul disco B: un nome diverso e cioè NVCRED.DAT.

Ricordate sempre che potete usare i caratteri jolly nei nomi di file per copiare più file.

Per esempio:

```
A>COPY *.BAT B:
```

Abbiamo così copiato tutti i file con estensione BAT da A: in B:.

COPY vi può inoltre servire per duplicare un file sullo stesso disco. Però

la copia deve avere un nome diverso perché l'MSX-DOS non permette che due file sullo stesso disco abbiano nomi uguali.

Per realizzare una copia del file CLIENTI.TXT date il comando:

```
A>COPY CLIENTI.TXT CLIENTI.NEW
          1 file copied
```

Potete notare che il disco di destinazione non è stato specificato per far restare la copia sullo stesso disco. La nuova copia di CLIENTI.TXT ha il nome CLIENTI.NEW.

Se state copiando da un disco generico al disco di default senza cambiare i nomi dei file, non avete bisogno di specificare il nome del disco di destinazione.

Per esempio, se A: è il drive di default e volete copiare il file CASSA.RPT da B: in A: date il comando

```
A>COPY B:CASSA.RPT
```

Potete usare il comando COPY quando volete che l'MSX-DOS legga o scriva sulle periferiche. Per far ciò, usate il nome della periferica al posto dei nomi dei file. I nomi delle periferiche sono stati esaminati nel Capitolo 18. In generale, però, il comando COPY non viene utilizzato per questo tipo di operazioni.

Se volete mandare alla stampante il file DATI.FOR, date il comando

```
A>COPY DATI.FOR PRN
```

Se definite come *sorgente* una periferica, il comando COPY continuerà a leggere dalla periferica finché non batterete i tasti CTRL Z seguiti da RETURN (CTRL Z è detto segnalatore di *end-of-file* cioè della fine del file). Questo metodo di individuazione della fine del file può sembrare un tantino arbitrario, ma è l'unico possibile.

Se volete inviare direttamente il testo nel file SHORT.MEM dovete usare il seguente comando:

```
A>COPY CON: SHORT.MEM
```

Il nome CON è il nome di periferica che rappresenta la *console*, cioè la tastiera e lo schermo. Dal momento in cui date il comando in poi, tutto ciò che battete sulla tastiera verrà messo direttamente nel file

senza che riappaia il prompt A>. Dopo aver battuto ciò di cui avete bisogno, dovete premere i tasti CTRL Z e RETURN.

Uso di COPY per concatenare i file

La seconda sintassi generale del comando è

COPY *sorgente1+sorgente2 destinazione*

Questa forma è usata per sommare file l'uno all'estremità dell'altro o, come si dice, per concatenarli. Potete concatenare nel file di destinazione quanti file volete.

Per esempio se volete realizzare un file di nome GRUPPO.TXT che sia una copia di GIANNA.TXT seguita da una copia di SILVIA.TXT, date il comando

```
A>COPY GIANNA.TXT + SILVIA.TXT GRUPPO.TXT
          1 file copied
```

Ora il file GRUPPO.TXT contiene GIANNA.TXT seguito da SILVIA.TXT.

Uso dell'argomento /V

Se l'MSX-DOS copia in modo sbagliato i vostri dati, può provocare danni molto gravi. Una copiatura scorretta è molto rara, tuttavia la maggior parte degli utenti preferisce essere al sicuro da tali inconvenienti. Per far ciò dovreste usare sempre l'argomento /V dopo un comando COPY.
In questo modo il comando sarà più lento ma vi assicurerete l'accuratezza delle copie. In ogni caso occorrerebbe più tempo per rileggere quello che l'MSX-DOS ha scritto. L'argomento /V si mette alla fine del comando COPY.

Per esempio questo comando:

```
A>COPY ACQUISTI.CON B: /V
          1 file copied
```

assicura che l'MSX-DOS copi accuratamente il file ACQUISTI.CON nel disco B:.

Altri usi

Altri due argomenti, /A e /B, vengono usati qualche volta con il comando COPY (anche se creano abbastanza confusione) per distinguere tra file ASCII e file binari (per questo si usano le lettere A e B). Mentre un file ASCII contiene solo testi normali, come una lettera o una nota, un file binario contiene caratteri speciali. I programmi e i data base sono di solito file binari. Gli argomenti /A e /B vengono usati solo per copiare file che si accordano strettamente a queste definizioni.

Regole

Non potete copiare un file su se stesso. Se cercate di copiare un file sullo stesso disco senza cambiarne il nome, l'MSX-DOS non eseguirà il comando.

Avvertimenti ed errori comuni

È molto facile invertire l'ordine dei file *sorgente* e *destinazione*.
Se cercate di copiare un file inesistente, l'MSX-DOS vi darà un messaggio di errore.
Ricordate di dare l'opzione /V, se copiate file, in modo da non rischiare di realizzare copie scorrette.
Prima di utilizzare una periferica come *sorgente* assicuratevi che la periferica stessa sia avviata correttamente. Lo stesso discorso vale soprattutto quando si tratta della porta di comunicazione. Ricordate infatti che in questi casi, il comando COPY aspetta sempre che i dati gli vengano forniti dalla periferica; una volta che i dati iniziano a passare, il comando continuerà a leggere sino alla fine del file. Se, per caso, date il nome di una periferica non abilitata a rispondervi, dovete resettare il computer.

Sintassi

La sintassi del comando può sembrare complicata, ma in realtà è molto facile da usare perché gli argomenti /A e /B non vengono utilizzati quasi mai. Vi mostriamo in successione varianti diverse dalla sintassi del comando che occupa, comunque, (come i comandi MSX-BASIC) sempre una linea.
Copia usuale di un file:

> COPY [{ / A| / B}] *specfile1* [{ / A| / B}]
> {*d:*|*specfile2*} [{ / A| / B}] [| / V]

Concatenamento di un file:

COPY [{ / A| / B} *specfile1a* [{ / A| / B}]
[+*specfile1b*] [{ / A| / B}]...[/ V]
{*d*:|*specfile2*} [{ / A| / B}] [/ V]

Copia in o da una periferica:

COPY {*specfile1*|*perif1*} {*specfile2*|*perif2*}

In ogni caso *specfile1* è la sorgente, mentre *specfile2* è la destinazione e il comando copia sempre il primo argomento al secondo. Potete anche usare nei nomi dei file i caratteri jolly che vi permettono di copiare più file contemporaneamente.
{ / A| / B} indica che potete usare / A o / B con qualsiasi specificatore di file. Il significato di / A o / B varia a seconda che li usiate come sorgente o destinazione. Possono apparire sia prima sia dopo lo specificatore della sorgente.
L'opzione / V fa sì che il comando verifichi l'accuratezza della copia che viene effettuata.
L'argomento *perif* rappresenta il nome di una qualsiasi delle periferiche (vedi il Capitolo 18 per periferiche e loro nomi).

19.6 Output dei file

Questa sezione esamina il comando TYPE.

TYPE

Questo comando visualizza un file sullo schermo.

Usi comuni

Il comando è molto facile da usare, dovete semplicemente battere la parola TYPE seguita dal nome del file che volete visualizzare:

TYPE *specfile*

L'MSX-DOS stampa così il file sullo schermo.

Per esempio, per vedere sullo schermo il contenuto del file RAPPORTO.TXT, date il comando

```
A>TYPE RAPPORTO.TXT
```

Questo comando è di tipo sequenziale; ciò significa che il suo utilizzo implica un'analisi del file dall'inizio alla fine: non potete tornare indietro e rileggere una porzione di file né muovervi liberamente all'interno del file per esaminare le informazioni contenute.
I programmi di editor oppure i word processor possono offrire, da questo punto di vista, una maggiore flessibilità: per questo motivo è opportuno usare un word processor per leggere un file, a meno che il file non sia corto o vogliate ottenere una copia su stampante.
Il comando TYPE, inoltre, non vi dà la possibilità di leggere programmi o altri file binari. File di questo tipo, che non possono essere visualizzati, hanno di solito estensioni COM, EXE, OVR o BIN; possono contenere tutti i 256 caratteri possibili che possono apparire sullo schermo come caratteri grafici oppure possono non apparire affatto. In ogni caso, una semplice lettura non vi dà la possibilità di utilizzare l'informazione contenuta in tali file.
Il comando TYPE fa un continuo scroll del file (cioè mostra il file riga per riga senza interruzioni) sul video. Perciò se il file è più lungo delle 24 o 25 righe disponibili sullo schermo, si muoverà tanto velocemente da impedirvi un'agevole lettura. Per leggere il file, potete allora usare la combinazione di tasti CTRL S per bloccare il testo sul video e CTRL Q per farlo procedere nuovamente. Per annullare il comando TYPE usate la combinazione CTRL C e tornerete all'MSX-DOS.
Se il file che state visualizzando ha caratteri tab (cioè caratteri generati usando il tasto TAB), il comando TYPE si posizionerà sulla prima colonna dello schermo che sia multiplo di 8 (cioè la colonna 8 o 16 o 24 e così via) e stamperà quindi ciò che segue il carattere tab.

Avvertimenti ed errori comuni

Il comando TYPE non vi impedisce la visualizzazione di file binari o di programmi: sappiate però che in questi casi, ciò che vedrete sarà, probabilmente, incomprensibile.

Sintassi

La sintassi del comando è

TYPE *specfile*

L'argomento *specfile* rappresenta lo specificatore del file da visualizzare e non deve contenere caratteri jolly.

19.7 Comandi per la gestione dei dischi

In questo paragrafo tratteremo i comandi DIR e FORMAT.

DIR

Questo comando elenca i file presenti su un disco e fornisce informazioni sulle loro dimensioni e il loro ultimo aggiornamento.

Usi comuni

In genere si richiede l'elenco di tutti i file presenti sul drive di default. La sintassi generale è

DIR [*d*:]

Per esempio la lista dei file contenuti sul drive A: può essere:

```
A>DIR
MSXDOS    SYS     2432  1-01-84
COMMAND   COM     6528 21-02-85
LOGO      TXT      117 10-12-85
TESTO1    TXT      755 15-10-85
SEC       COM      117 10-12-85
TEMPO     TXT      460 10-12-85
TESTO     BAK      755 15-10-85
TESTO     TXT      755 15-10-85
TESTO2    TXT      755 15-10-85
RPTCOPY   BAT       51 12-12-85
       10 files   332800 bytes free
A>
```

L'elenco visualizzato da DIR contiene molte specifiche:

- Le prime due colonne dell'elenco contengono il nome dei file e la loro estensione che, in questo caso, non viene separata da un punto.
- La terza colonna contiene la dimensione del file in byte.
- L'ultima colonna fornisce la data in cui ogni file è stato aggiornato per l'ultima volta; se non lo è mai stato, queste informazioni riguardano il

momento in cui il file è stato generato (la data può essere ovviamente sbagliata, se il vostro computer non ha un calendario perpetuo o non vi siete preoccupati di indicare la data corretta all'accensione).

- L'ultima riga riassume quanti file sono nell'elenco e quanti byte restano liberi nel disco per altri file.
Specificando il nome di un altro disco potete visualizzarne il directory.

Per esempio il comando

```
A>DIR B:
```

visualizza sullo schermo la lista del directory del disco B:.

Uso di DIR su singoli file

Il comando DIR permette di ottenere anche informazioni su singoli file o su un certo gruppo di file mediante l'uso dei caratteri jolly.

Per esempio, per avere informazioni sul file LOGO.TXT basterà indicarne lo specificatore dopo il comando nel modo seguente:

```
A>DIR LOGO.TXT
LOGO      TXT      117 10-12-85
        1 file   352256 bytes free
```

Se volete invece ottenere l'elenco di tutti i file con estensione TXT date il comando:

```
A>DIR *
LOGO      TXT      117 10-12-85
TESTO1    TXT      755 15-10-85
TEMPO     TXT      460 10-12-85
TESTO     TXT      755 15-10-85
TESTO2    TXT      755 15-10-85
        5 files  352256 bytes free
```

L'uso di un carattere jolly (*) sia per il nome sia per l'estensione è talmente comune che il comando DIR vi permette di ometterlo.

Per esempio, il comando precedente (DIR *.TXT) può anche essere

```
A>DIR .TXT
```

Il carattere jolly è stato omesso, però è necessario battere il punto prima dell'estensione perché, altrimenti, l'MSX-DOS lo scambierebbe per il nome del file.

Uso degli argomenti per modificare l'elenco

Esistono due argomenti che possono essere usati con il comando DIR: / W e / P. Se il catalogo contiene un gran numero di file e non vi interessa conoscerne la dimensione e la data di aggiornamento, potete ottenere un elenco semplificato usando l'argomento / W.

Notate per esempio, la differenza fra i due cataloghi seguenti:

```
A>DIR
MSXDOS    SYS    1024  1-01-84
COMMAND   COM    1024 21-02-85
LOGO      TXT     117 10-12-85
TESTO1    TXT     755 15-10-85
SEC       COM     117 10-12-85
TEMPO     TXT     460 10-12-85
TESTO     BAK     755 15-10-85
TESTO     TXT     755 15-10-85
TESTO2    TXT     755 15-10-85
RPTCOPY   BAT      51 12-12-85
       10 files   352256 bytes free

A>DIR/W
MSXDOS    SYS  COMMAND  COM
LOGO      TXT    TESTO1   TXT
SEC       COM    TEMPO    TXT
TESTO     BAK    TESTO    TXT
TESTO2    TXT    RPTCOPY  BAT
       10 files   352256 bytes free
```

L'argomento / P fa sì che l'elenco si fermi alla fine di ogni pagina. Contemporaneamente, apparirà in fondo allo schermo il messaggio "Strike a key when ready...", che vi invita a premere un tasto se volete visualizzare la pagina seguente del catalogo.

Sintassi

La sintassi del comando DIR è

DIR [(*d*:|*specfile*)] [/ P] [/ W]

Se non usate argomenti il programma produrrà l'elenco dei file del disco di default. Se specificate un nome di drive (*d*:) vi verrà mostrato il catalogo di quel particolare disco; se darete uno *specfile* (anche usando caratteri jolly) otterrete l'elenco di quei soli file che avete specificato.

L'argomento /W mostra, su quattro colonne, l'elenco dei file senza dimensione e data. L'argomento /P fa sì che l'elenco venga mostrato con pause alla fine di ogni pagina.

FORMAT

Il comando FORMAT prepara un dischetto per l'uso con l'MSX-DOS.

Usi comuni

In genere, un dischetto al momento dell'acquisto non è formattato; dovete perciò, prima di scrivere sul dischetto, usare il comando FORMAT. Dovete prestare molta attenzione quando usate questo comando, perché uno dei suoi effetti è quello di cancellare tutte le informazioni che il disco contiene.
La sua sintassi è molto semplice:

FORMAT

Avvertimenti ed errori comuni

Ricordate che il comando FORMAT cancella tutte le informazioni presenti sul disco; fate perciò attenzione quando lo usate su un disco già scritto: accertatevi che le informazioni in esso contenuto non vi interessino più.

Sintassi

La sintassi del comando è

FORMAT [*d*:]

Lo specificatore di drive va indicato se il disco da formattare non è inserito nel drive di default.

19.8 Comandi per la messa a punto del sistema

Questo paragrafo tratta i comandi DATE, TIME e MODE. Questi comandi vengono usati per mettere a punto le varie *system setting* dell'MSX-DOS,

cioè quelle informazioni interne dalle quali il sistema riconosce le condizioni in cui volete che esso operi. Non è fondamentale conoscere questi comandi ma il loro apprendimento e il loro uso sono abbastanza semplici. Ogni volta che caricate l'MSX-DOS il sistema dà valori standard a molte variabili. Se queste inizializzazioni non vi sembrano opportune o adatte, potete cambiarle con i comandi presentati qui di seguito. Per cambiarle automaticamente includete i comandi appropriati nel file AUTOEXEC.BAT.

DATE

Il comando DATE inizializza la data usata dall'MSX-DOS quando aggiorna i file. Il comando agisce come la richiesta della data che vedete all'atto del caricamento del sistema.

Usi comuni

Potete indicare la nuova data sulla stessa riga del comando oppure lasciare che il comando stesso ve la richieda.
La sintassi generale è

DATE [*gg-mm-aa*]

Con il comando si modifica la locazione della RAM che contiene la data e non la data del file che viene assegnata al momento della scrittura su disco.

Per esempio potete dare la nuova data nel modo seguente:

```
A>DATE 7/10/84
```

Per ottenere il messaggio da parte del comando DATE, scrivete

```
A>DATE
Current date is Mon 12-10-1984
Enter new date:
```

Se la data segnalata risultasse esatta e non ci fosse bisogno di modificarla, basterà premere il tasto RETURN e nulla verrà cambiato.
Alcuni computer dispongono di un orologio interno che può essere letto da un programma che inizializza automaticamente la data e l'ora senza che lo dobbiate fare voi manualmente.

Regole

La data da battere deve essere composta da tre numeri separati da trattini (-) oppure da barre (/). Le regole esatte sono già state fornite nel paragrafo 18.1.

Avvertimenti ed errori comuni

Alcuni programmi BASIC cambiano la data senza che ve ne accorgiate: se scoprite che la data non è stata inserita correttamente, dopo l'esecuzione di un programma potete aggiornarla con il comando DATE.

Sintassi

La sintassi del comando DATE è

DATE [*gg-mm-aa*]

dove *gg-mm-aa* stanno per il giorno, il mese e l'anno. Questi parametri sono separati da trattini (-) o barre (/) e se non sono indicati esplicitamente dopo il comando, il comando stesso ve li chiederà.

TIME

Il comando TIME inizializza l'ora usata dall'MSX-DOS quando aggiorna i file e ha lo stesso meccanismo della richiesta dell'ora che appare quando caricate l'MSX-DOS. Il comando TIME non è attivo su molti computer MSX.

Usi comuni

La sintassi generale del comando è

TIME [*hh:mm:ss.xx*]

L'ora può essere indicata esplicitamente dopo il comando oppure il comando ve la richiederà.

Per esempio potete indicare subito l'ora modificata

```
A>TIME 9:41
```

oppure dare solo il comando TIME e ottenere così il messaggio di richiesta

```
A>TIME
Current time is 9:41:03
Enter new time:
```

Se non aveste bisogno di modificare l'ora segnalata basta premere RETURN.
Esistono computer dotati di orologio interno ed è possibile, in questi casi, che l'ora e la data vengano letti da un programma e inizializzati automaticamente.

Regole

L'unico formato corretto consiste nell'esprimere l'ora in ore, minuti e secondi separati dai due punti (:).

Avvertimenti ed errori comuni

Alcuni programmi BASIC modificano l'ora per calcolare la durata di determinati avvenimenti azzerandola prima dell'uso. Perciò se scoprite che, dopo aver eseguito un programma, l'ora è sbagliata, dovete utilizzare il comando TIME.

Sintassi

La sintassi del comando TIME è

TIME [*hh:mm:ss.xx*]

dove *hh:mm:ss.xx* sta a significare ore, minuti, secondi e centesimi di secondo. Dovete separare le ore dai minuti e dai secondi con i due punti e i secondi dai centesimi di secondo con il punto. L'indicazione dei secondi e dei centesimi di secondo è opzionale.

MODE

Il comando MODE assegna il numero di caratteri per ogni riga dello schermo, come il comando WIDTH nell'MSX-BASIC.

Usi comuni

È improbabile che vogliate modificare la larghezza dello schermo, dato che è preferibile usare sempre il massimo numero di colonne possibili. La sintassi generale è

MODE *larghezza*

larghezza va da 1 a 40.

19.9 Comandi per file batch

In questa sezione presenteremo i comandi PAUSE e REM. Questi comandi si usano solo all'interno di file batch.
Nel Capitolo 18 abbiamo mostrato come costruire i file batch che sono, in breve, file che contengono una lista comprendente comandi MSX-DOS, nomi di programmi applicativi o di altri file batch. In particolare si è già parlato del file AUTOEXEC.BAT che viene eseguito automaticamente quando si carica il sistema.
Ora però conoscete molti più comandi che potete inserire nel file AUTOEXEC.BAT.

PAUSE

Quando usate il comando PAUSE vedete comparire il messaggio "Strike a key when ready...". Ciò significa che il sistema vi chiede di premere un tasto quando siete pronti ad effettuare una determinata operazione a poi attende, prima di continuare, la vostra risposta.

Usi comuni

È un comando utile se le operazioni che volete eseguire richiedono un'attivazione manuale.

Supponiamo che il compito di un vostro file batch PAGREG.BAT sia quello di visualizzare e poi stampare il file PR1.DAT e vogliate prima chiedere all'utente di verificare che la stampante sia accesa. Il file batch contiene:

PAGREG.BAT

```
TYPE PR1.DAT
PAUSE Controllare la stampante
COPY PR1.DAT PRN
```

Questo file batch vi permette quindi di controllare se la stampante è accesa prima di inviarvi un file. Mentre il comando PAUSE è attivo, potete fare qualsiasi cosa salvo usare la tastiera (altrimenti il comando terminerebbe e il sistema eseguirebbe l'azione successiva).

Il comando PAUSE viene usato comunemente anche per cambiare un dischetto sul drive.

Sintassi

Il comando PAUSE ha la seguente sintassi:

PAUSE [*messaggio*]

e deve essere incluso in un file batch. In *messaggio* potete inserire un vostro commento.

REM

Il comando REM serve ad aggiungere commenti ad un file batch e non produce alcun risultato. Ha la stessa funzione del comando REM in MSX-BASIC.

Usi comuni

Il comando vi permette di annotare nel file stesso informazioni utili come, per esempio, il suo contenuto oppure il compito da esso svolto.

È sempre utile aggiungere a un file batch una riga di commento che vi ricordi le ragioni per cui è stato scritto.

Sintassi

La sintassi del comando REM è

REM *commento*

commento è un testo qualsiasi. Il comando può essere messo ovunque in un file batch.

Appendice

Guida rapida MSX-BASIC

In questa appendice sono elencati in ordine alfabetico tutti i comandi e le istruzioni (A.1) e le funzioni MSX-BASIC (A.2), con la sintassi ed una breve spiegazione del loro uso.

A.1 Comandi e istruzioni MSX-BASIC

AUTO

Genera automaticamente il numero di linea successivo ogni volta che viene premuto il tasto RETURN

Sintassi

AUTO [*inizio*] [,*incremento*]

inizio è il numero di linea da cui partire per la numerazione automatica, *incremento* è il valore che deve essere aggiunto ad ogni numero di linea per ottenere il successivo. Se uno o entrambi i valori vengono omessi, viene posto come valore di default 10.

BEEP

Genera un semplice beep

Sintassi
BEEP

BLOAD

Carica in memoria un file binario

Sintassi
BLOAD "*nomefile*" [,R] [,S] [,*offset*]

Con l'opzione R un programma in codice macchina viene caricato ed eseguito; l'opzione S si usa per caricare un'immagine dalla VRAM; se viene specificato *offset* il suo valore viene aggiunto agli indirizzi finale e iniziale.

BSAVE

Salva un'area di memoria in un file binario

Sintassi
BSAVE "*nomefile*", *indiniz*, *indfin*

nomefile è il nome del file nel quale si vuole immagazzinare l'immagine binaria, *indiniz* e *indifin* sono gli indirizzi di inizio e fine della parte di memoria che volete salvare.

CALL

Chiama una routine in codice macchina

Sintassi
CALL *routine* (*param1*,*param2*,...)
_*routine*(*param1*,*param2*,...)

routine è il nome del programma nella ROM; eventuali argomenti vanno indicati in parentesi. È usato prevalentemente per formattare i dischi e per passare da ambiente BASIC a ambiente DOS e viceversa:
CALL FORMAT
CALL BASIC
CALL SYSTEM

CIRCLE

Disegna in modo grafico linee curve, cerchi ed ellissi con centro nel punto di coordinate (*x*, *y*) e col *raggio* indicato

Sintassi

CIRCLE [STEP] (*x*,*y*),*raggio* [,[*colore*], [*angoloiniziale*], [*angolofinale*] [,*rapporto*]]

Con l'argomento STEP il centro è relativo alla corrente posizione del cursore grafico; *colore* è un numero compreso tra 1 e 15 che specifica il colore in cui verrà disegnato il cerchio (per default colore di primo piano). Se vengono specificati *angoloiniziale* e *angolofinale* (in radianti), verrà disegnata la porzione di arco circolare con gli estremi indicati.
rapporto è un'espressione numerica che indica il rapporto tra il raggio orizzontale e quello verticale (default 1).

CLEAR

Azzera tutte le variabili in memoria, stabilisce le dimensioni dell'area disponibile per le variabili stringa e l'indirizzo massimo di memoria usata dal BASIC

Sintassi

CLEAR [*dimstringhe*] [,*indmax*]

dimstringhe è per default 200.
CLEAR chiude tutti i file.

CLOAD

Carica un programma da cassetta

Sintassi

CLOAD ["*nomefile*"]

nomefile è il nome del programma da caricare, composto al massimo di 6 caratteri. Se viene omesso, viene caricato il primo programma trovato.

CLOSE

Chiude un file

Sintassi
CLOSE [[#] *numfile1*] [,*numfile2*]...

numfile è il numero assegnato al file all'apertura. Se non vengono indicati quelli da chiudere, vengono chiusi tutti i file.

CLS

Cancella lo schermo

Sintassi
CLS

COLOR

Definisce il colore di primo piano, dello sfondo e del bordo

Sintassi
COLOR [*primopiano*],[*sfondo*],[*bordo*]

I tre parametri possono assumere valori da 0 a 15, a seconda del colore prescelto.

CONT

Fa riprendere l'esecuzione di un programma interrotto con CTRL STOP

Sintassi
CONT

COPY

Copia un file

Sintassi
COPY "*file1*" TO "*file2*"

file1 è il nome del file che volete copiare e *file2* è il nome da assegnare alla copia del file.

CSAVE

Memorizza un programma su cassetta

Sintassi

CSAVE "*nomefile*" [,*baud*]

nomefile è il programma che si vuol memorizzare, composto da un massimo di 6 caratteri; *baud* indica la velocità di trasferimento dei caratteri e può essere 1 (=1200 baud, valore di default) o 2 (=2400 baud).

DATA

Contiene i dati numerici o di tipo stringa utilizzati dalle istruzioni READ

Sintassi

DATA *elemento1*[,*elemento2*],...

DEF

Definisce il tipo di una variabile

Sintassi

DEF *tipo lettera*[–*lettera*] [,*lettera*[–*lettera*]]...

tipo può essere INT (variabile intera), SNG (variabile in precisione semplice), DBL (variabile in doppia precisione), STR (variabile a stringa). DEF dichiara che tutte le variabili il cui nome inizia con una lettera compresa nell'intervallo *lettera–lettera* sono variabili del *tipo* indicato.

DEF FN

Definisce una funzione scritta dall'utente

Sintassi

DEF FN *nome* [(*argomento*[,*argomento*]...)]=*espressione*

nome deve essere un nome di variabile valido che diventa, preceduto da FN, il nome della funzione; *argomento* è un nome di variabile, che compare in *espressione*, che deve essere sostituito da un valore ogni volta che la funzione viene richiamata. *espressione* definisce il valore restituito dalla funzione ed il suo tipo deve essere uguale al tipo di *nome*.

DEFUSR

Specifica l'indirizzo di partenza di una subroutine in codice macchina che verrà chiamata con la funzione predefinita USR

Sintassi

DEFUSR[*n*]=*ind*

n è un numero da 0 a 9 (default=0) che identifica la routine USR il cui indirizzo viene specificato in *ind*.

DELETE

Cancella linee di programma

Sintassi

DELETE [*linea1*] [–*linea2*]
DELETE [*linea1*–]

linea1 è il numero della prima linea da cancellare; *linea2* è il numero dell'ultima linea da cancellare; il trattino sostituisce la prima o l'ultima linea del programma (nella prima e nella seconda versione sintattica, rispettivamente).

DIM

Specifica il valore massimo per gli indici di un array

Sintassi

DIM *nome*(*indice*[,*indice*]...)

nome è il nome che viene attribuito alla variabile array; *indice* è un'espressione numerica che definisce il valore massimo di una dimensione dell'array.

DRAW

Esegue un disegno in modo grafico

Sintassi
DRAW "*stringa*"

stringa è un insieme di comandi grafici e di costanti numeriche, ciascuno separato da virgole, che definiscono il disegno da eseguire (per la sintassi di ciascun comando vedi Figura 11.7).

END

Termina l'esecuzione di un programma e chiude i file

Sintassi
END

ERASE

Cancella gli array specificati

Sintassi
ERASE *nomearray*[,*nomearray*]...

nomearray è il nome dell'array che si desidera cancellare.

ERROR

Genera un errore corrispondente al numero specificato o definisce un numero d'errore

Sintassi
ERROR *n*

n è un numero intero da 0 a 59 nel caso di generazione dell'errore, da 60 a 255 nel caso della definizione di un nuovo errore.

FIELD

Riserva spazio per le variabili in un buffer per file ad accesso diretto

Sintassi

FIELD [#] *numfile, larg1* AS *var$1* [,*larg2* AS *var$2*]...

numfile è il numero con cui il file è stato aperto; *larg* (valore massimo 255) specifica il numero di posizioni per i caratteri da riservare per *var$*.

FILES

Visualizza i nomi dei file residenti su disco

Sintassi

FILES ["*specfile*"]
LFILES ["*specfile*"]

I file visualizzati sono tutti quelli presenti sul disco se *specfile* non è indicato; in caso contrario, sono quelli il cui nome corrisponde a *specfile* che può contenere i caratteri jolly "*" e "?".
Nella forma LFILES il catalogo viene inviato alla stampante.

FOR NEXT

Esegue una serie di istruzioni per un dato numero di volte

Sintassi

FOR *variabile* = *inizio* TO *fine* [STEP *passo*]
.
.
.
NEXT [*variabile*] [,*variabile*]...

inizio è il valore iniziale assegnato al contatore *variabile*; dopo la prima esecuzione delle istruzioni comprese tra la linea FOR e la linea NEXT, *variabile* viene incrementato della quantità specificata da *passo* (default=1) e l'esecuzione viene ripetuta fino a quando *variabile* ha superato il valore di *fine*. Se *inizio*>*fine*, il contatore viene decrementato e *passo* deve venir specificato. Se più loop FOR NEXT sono annidati, un'unica linea NEXT può venir usata per chiudere tutti i loop.

GET

Legge un record da un file

Sintassi
GET #*n*, *numrec*

n è il numero con cui il file è stato aperto; *numrec* è il numero di record da leggere.

GOSUB

Trasferisce il controllo ad una subroutine

Sintassi
GOSUB *linea*
.
.
.
RETURN

linea è il numero della prima linea della subroutine. L'istruzione RETURN indica la fine della subroutine e il ritorno dell'esecuzione del programma all'istruzione che segue GOSUB.

GOTO

Compie un salto incondizionato ad uno specificato numero di linea

Sintassi
GOTO *linea*

IF THEN ELSE

Prende una decisione riguardante l'andamento del programma basandosi sul risultato di una espressione

Sintassi
IF *espressione* THEN *proposizione* [ELSE *proposizione*]

proposizione può essere un'istruzione, una sequenza di istruzioni o un numero di linea. Se *espressione* è vera viene eseguita la *proposi-*

zione che segue THEN, se è falsa viene eseguita la *proposizione* che segue ELSE, se presente, o la prima istruzione successiva.

INPUT

Riceve un input da tastiera

Sintassi
INPUT ["*prompt*"]; *variabile*[,*variabile*]

prompt è una stringa di richiesta dati che viene eventualmente visualizzata prima del punto interrogativo. *variabile* è il nome di una variabile numerica, a stringa o array che ha la funzione di ricevere l'input.

INPUT

Legge dati da un file e li assegna alle variabili del programma

Sintassi
INPUT #*n*, *variabile*[,*variabile*]

n è il numero assegnato al file al momento dell'apertura. *variabile* può essere una variabile numerica, stringa o array.

INTERVAL

Attiva, disattiva o sospende un'interruzione specificata da ON INTERVAL GOSUB

Sintassi
INTERVAL ON
INTERVAL OFF
INTERVAL STOP

KEY

Assegna o visualizza la funzione dei tasti definibili

Sintassi

KEY *n, "stringa"*
KEY LIST
KEY OFF
KEY ON

n è il numero del tasto funzione (da 1 a 10). *stringa* (massimo 15 caratteri) è l'espressione che sarà assegnata al tasto funzione. KEY LIST visualizza l'elenco dei valori correnti dei tasti; KEY OFF disabilita la visualizzazione dei tasti funzione sull'ultima riga di schermo; KEY ON la riattiva.

KEY (n)

Attiva, disattiva o sospende l'intercettamento di un tasto funzione, specificato da ON KEY GOSUB

Sintassi

KEY (*n*) ON
KEY (*n*) OFF
KEY (*n*) STOP

n è il tasto funzione da intercettare.

KILL

Cancella file da disco

Sintassi

KILL "*nomefile*[''']

nomefile è il nome del file che si vuole cancellare.

LET

Assegna ad una variabile il valore di un'espressione

Sintassi

[LET] *variabile = espressione*

variabile è il nome di una variabile o elemento di array che deve ricevere il valore di *espressione*.

LINE INPUT

Legge da tastiera una stringa composta al massimo da 254 caratteri e l'assegna a una variabile

Sintassi
LINE INPUT ["*prompt*"]; *variabile$*

prompt è una stringa di richiesta dati; *variabile$* è una variabile stringa. LINE INPUT non visualizza il punto interrogativo e non riconosce i delimitatori.

LINE INPUT

Legge un'intera linea (fino a 254 caratteri) da un file e l'assegna a una variabile

Sintassi
LINE INPUT #*numfile, variabile$*

numfile è il numero con il quale il file è stato aperto; *variabile$* è la variabile a stringa a cui la stringa viene assegnata.

LINE

Disegna linee rette o rettangoli in modo grafico

Sintassi
LINE [[STEP] (*x1, y1*)] – [STEP] (*x2,y2*)[,*colore*] [,B [F]]

(*x1,y1*) e (*x2,y2*) sono le coordinate di inizio e fine della linea da tracciare; se non viene specificato (*x1,y1*) viene presa per default l'ultima posizione specificata da una istruzione grafica. *colore* (da 1 a 15) seleziona il colore del disegno. B disegna un rettangolo che ha come diagonale la linea retta specificata, BF disegna lo stesso rettangolo e lo riempie di colore. STEP rende l'origine delle coordinate relativa all'ultimo punto specificato.

LIST

Visualizza il listato del programma corrente in memoria

Sintassi
LIST [*linea1*] [– *linea2*]
LLIST [*linea1*] [– *linea2*]

linea1 è la prima linea da visualizzare e *linea2* l'ultima (per default la prima e l'ultima linea del programma). In assenza di parametri viene visualizzato tutto il listato.
Nella forma LLIST il listato viene inviato alla stampante.

LOAD

Carica in memoria un programma

Sintassi
LOAD "*specfile*["]

specfile può contenere il nome del dispositivo da cui caricare il programma (ad esempio CAS:).

LOCATE

Posiziona il cursore

Sintassi
LOCATE *x*, *y*

x (da 0 a 39) indica la colonna, *y* (da 0 a 24) la riga in cui si vuole visualizzare il cursore.

MAXFILES=

Specifica il numero massimo di file che possono essere aperti

Sintassi
MAXFILES= *n*

MERGE

Carica un programma e lo fonde con quello corrente in memoria

Sintassi
MERGE "*nomefile*"

nomefile è il nome del programma che si vuole caricare e fondere e deve essere stato precedentemente salvato in formato ASCII.

MOTOR

Spegne e accende il registratore a cassette

Sintassi
MOTOR ON
MOTOR OFF

NAME

Cambia il nome di un file su disco

Sintassi
NAME "*vecchionome*" AS "*nuovonome*"

vecchionome è un nome di file presente sul disco.

NEW

Cancella dalla memoria il programma corrente

Sintassi

NEW

ON ERROR GOTO

Trasferisce l'esecuzione del programma a un dato numero di linea quando si verifica un errore

Sintassi
ON ERROR GOTO *linea*

linea è il numero di linea a cui si vuole trasferire l'esecuzione del programma.

ON INTERVAL GOSUB

Definisce il numero di linea in cui inizia la subroutine di intercettamento di un'interruzione provocata da un timer interno

Sintassi

ON INTERVAL = *n* GOSUB *linea*

n è l'intervallo di tempo espresso in cinquantesimi di secondo.

ON KEY GOSUB

Definisce il numero di linea in cui inizia la subroutine di intercettamento di un tasto funzione.

Sintassi

ON KEY GOSUB *linea* [,*linea*]...

Dopo GOSUB possono essere specificati fino a 5 numeri di linea in corrispondenza dei 5 tasti funzione.

ON GOSUB e ON GOTO

Trasferiscono l'esecuzione ad uno dei numeri di linea indicati secondo il valore di un'espressione

Sintassi

ON *espressione* GOSUB *linea* [,*linea*]
ON *espressione* GOTO *linea* [,*linea*]

Se il valore di *espressione* (che deve essere compreso tra 0 e 255) è 1 viene eseguita la prima *linea*, se è 2 la seconda e così via. Se è 0 o maggiore del numero di elementi della lista l'esecuzione prosegue dalla prima istruzione eseguibile.

ON SPRITE GOSUB

Definisce il numero di linea in cui inizia la subroutine di intercettamento delle collisioni tra sprite

Sintassi
ON SPRITE GOSUB *linea*

ON STOP GOSUB

Definisce il numero di linea in cui inizia la subroutine di intercettamento di un'interruzione causata da CTRL STOP

Sintassi
ON STOP GOSUB *linea*

ON STRIG GOSUB

Definisce il numero di linea in cui inizia la subroutine di intercettamento della pressione di un pulsante del joystick o della barra di spazio

Sintassi
ON STRIG GOSUB *linea*[, *linea*]

Possono essere specificati fino a 5 numeri di *linea*.

OPEN

Apre i file

Sintassi
OPEN "*nomefile*" FOR *modo* AS [#]*n*
OPEN "*nomefile*" AS [#] *n* LEN=*lungrec*

nomefile è il nome del file che si vuole aprire (max 7 caratteri) o del dispositivo ed *n* è il numero che viene assegnato al file. La prima forma apre un file sequenziale. *modo* può essere: INPUT (per l'input sequenziale), OUTPUT (per l'output sequenziale), APPEND (per aggiungere dati alla fine del file).
La seconda forma apre un file ad accesso casuale dove *lungrec* assegna la lunghezza dei record.

OUT

Invia un byte verso la porta specificata

Sintassi
OUT *porta, dato*

porta è l'indirizzo di destinazione del *dato* che è un intero da 0 a 255.

PAINT

Colora un'area

Sintassi
PAINT [STEP] *(x,y)* [,*riempimento*] [,*contorno*]

(x,y) sono le coordinate di un punto all'interno dell'area da colorare (relative all'ultimo punto se è presente STEP); *riempimento* e *contorno* sono i colori (da 1 a 15) rispettivamente per la colorazione dell'area e per la linea di contorno. In modo SCREEN 2 devono essere identici al colore di primo piano, in SCREEN 3 possono essere diversi tra loro.

PLAY

Suona un brano musicale

Sintassi
PLAY *"stringa1","stringa2","stringa3"*

stringa1, stringa2, stringa3 sono insiemi di comandi musicali e costanti numeriche che definiscono i suoni da generare, rispettivamente per la voce 1, voce 2 e voce 3.

POKE

Scrive un byte in una locazione di memoria

Sintassi
POKE *ind, valore*

ind è l'indirizzo o locazione del byte della RAM che volete controllare (0–65535); *valore* è un numero tra 0 e 255 da porre nella locazione.

PSET e PRESET

Disegnano un punto in modo grafico

Sintassi
{PSET|PRESET} [STEP] (*x,y*) [,*colore*]

colore (da 1 a 15) è il colore in cui viene disegnato il punto (con PSET è per default quello di primo piano, con PRESET quello dello sfondo); (*x,y*) sono le coordinate in cui si vuole disegnare il punto (relative all'ultimo punto se presente STEP). Senza *colore* PSET disegna punti e PRESET li cancella; con *colore* le due istruzioni sono identiche.

PRINT

Visualizza dati sullo schermo

Sintassi
{PRINT|?} *lista di espressioni* [;] [,]

lista di espressioni è una lista di espressioni numeriche o a stringa o entrambe separate da virgole, o punto e virgola. Le stringhe devono essere racchiuse tra virgolette.

PRINT USING

Stampa stringhe o numeri usando uno speciale formato

Sintassi
PRINT USING "*stringaformato*"; *argomento1,argomento2,...*

stringaformato consiste di speciali caratteri di formattazione che determinano il campo e il formato dei dati da stampare contenuti in *argomento1, argomento2,* ecc.

PRINT# e PRINT# USING

Scrivono i dati in un file

Sintassi
PRINT#*n*,[USING "*stringaformato*";] *lista di espressioni*

n è il numero assegnato al file all'apertura; *stringaformato* è composta da caratteri di formato; *lista di espressioni* è un elenco delle espressioni numeriche o a stringa che devono essere scritte nel file.

PUT#

Scrive in un file ad accesso casuale

Sintassi
PUT [#] *n*, *numrec*

n è il numero con il quale il file è stato aperto; *numrec* è il numero del record che deve essere scritto.

PUT SPRITE

Visualizza uno sprite

Sintassi
PUT SPRITE *numerosprite*, [[STEP] (*x*,*y*)] [,*colore*] [,*immagine*]

numerosprite è il numero assegnato allo sprite (da 0 a 31); (*x*,*y*) sono le coordinate in cui si vuole visualizzare lo sprite (relative con STEP); *colore* è per default quello di primo piano, *immagine* è il numero con cui è stato memorizzato lo sprite (per default uguale a *numerosprite*).

READ

Legge i valori delle istruzioni DATA e li assegna alle variabili

Sintassi
READ *variabile* [, *variabile*]

REM

Inserisce commenti in un programma

Sintassi
{REM | '} *commento*

commento è una sequenza di caratteri.

RENUM

Rinumera le linee di programma

Sintassi
RENUM [*nuovonum*] [,[*vecchionum*] [,*incremento*]

nuovonum è il primo numero di linea che deve essere usato nella sequenza (per default 10); *vecchionum* è la linea da cui deve partire la rinumerazione (per default la prima); *incremento* è l'incremento da usare nella nuova sequenza (per default 10).

RESTORE

Consente di rileggere le istruzioni DATA

Sintassi
RESTORE [*linea*]

linea è il numero di linea dell'istruzione DATA nel programma cui accede READ (per default la prima istruzione DATA del programma).

RESUME

Riporta l'esecuzione al programma principale dopo una routine di intercettamento d'errore

Sintassi
RESUME[0]
RESUME NEXT
RESUME *linea*

RESUME o RESUME 0 riprende dall'istruzione che ha causato l'erro-

re. RESUME NEXT riprende da quella immediatamente seguente quella che ha provocato l'errore .RESUME *linea* va al numero di linea indicato.

RETURN

Conclude una subroutine e invia il controllo all'istruzione successiva a quella che aveva chiamato la subroutine

Sintassi

RETURN [*linea*]

linea è il numero della linea alla quale si vuol inviare il controllo (per default è l'istruzione successiva a quella che aveva chiamato la subroutine).

RUN

Avvia l'esecuzione di un programma

Sintassi

RUN [*linea*]

linea è il numero di linea da cui si vuole iniziare l'esecuzione del programma (per default la prima).

SAVE

Salva un programma su disco

Sintassi

SAVE "*nomefile*["][,A]

nomefile deve essere al massimo di 6 caratteri. L'opzione A salva il file in formato ASCII.

SCREEN

Definisce il modo di visualizzazione, le dimensioni dello sprite, il clic dei tasti, la frequenza di trasmissione per la registrazione dei dati su cassetta e il tipo di stampante

Sintassi

SCREEN [*modo*] [,*dimsprite*] [,*clic*] [,*baud*] [,*stamp*]

modo è 0, 1, 2 o 3 (modo testo 40×24, modo testo 32×24, modo grafico e modo a colori); *dimsprite* è 0, 1, 2 o 3; *clic* è 1 (clic attivato) o 0 (clic non attivato); *baud* è 1 (1200) o 2 (2400); *stamp* è 0 (stampante grafica) o diverso (stampante non grafica).

SOUND

Genera effetti sonori

Sintassi

SOUND *registro, valore*

registro è il numero di registro (da 0 a 13); *valore* è il numero da immagazzinare nel registro.

SPRITE

Attiva, disattiva o sospende un intercettamento di collisione di sprite

Sintassi

SPRITE ON
SPRITE OFF
SPRITE STOP

STOP

Sospende l'esecuzione del programma

Sintassi

STOP

STOP ON
STOP OFF
STOP STOP

Attiva, disattiva o sospende l'intercettamento dell'interruzione causata da CTRL STOP

STRIG

Attiva, disattiva o sospende l'intercettamento della pressione del pulsante del joystick o della barra di spazio

Sintassi
STRIG ON
STRIG OFF
STRIG STOP

SWAP

Scambia i valori di due variabili

Sintassi
SWAP *var1,var2*

TRON e TROFF

TRON visualizza passo-passo l'esecuzione del programma.
Con TROFF si annulla l'istruzione TRON

Sintassi
TRON
TROFF

VPOKE

Scrive un byte sulla memoria RAM video

Sintassi
VPOKE *n, byte*

n è l'indirizzo di memoria in cui si vuole scrivere *byte*.

WAIT

Attende che i dati provenienti da un canale I/O soddisfino determinate condizioni

Sintassi
WAIT *porta, esprand, esprxor*

I dati provenienti da *porta* vengono confrontati con *esprxor* (istruzione XOR); il risultato viene confrontato con *esprand* (istruzione AND).

WIDTH

Specifica il numero di colonne dello schermo in modo testo

Sintassi
WIDTH *n*

n va da 0 a 40 in modo 0 e da 0 a 32 in modo 1.

A.2 Funzioni MSX-BASIC

ABS

Restituisce il valore assoluto dei dati numerici

Sintassi
ABS(*n*)

Il valore assoluto di un numero è sempre positivo o zero.

ASC

Restituisce il valore in codice ASCII per il primo carattere di stringa

Sintassi
ASC("stringa")

ATN

Restituisce il valore dell'arcotangente di dati numerici

Sintassi
ATN(*n*)

Restituisce il valore dell'angolo la cui tangente è *n* espresso in radianti.

BASE

Legge o modifica l'indirizzo base usato dal chip VDP per accedere alla VRAM

Sintassi
BASE(*n*)
BASE(*n*) = *indvram*

n varia da 0 a 19.

BIN$

Restituisce l'espressione binaria a stringa di dati numerici

Sintassi
BIN$(*n*)

CDBL

Converte *n* in un numero in doppia precisione

Sintassi
CDBL(*n*)

CHR$

Converte un codice ASCII nel corrispondente carattere

Sintassi
CHR$(*n*)

n deve essere compreso tra 0 e 255

CINT

Converte dati numerici in interi

Sintassi
CINT(*n*)

Elimina tutte le cifre alla destra della virgola.

COS

Restituisce il coseno di *angolo*

Sintassi
COS (*angolo*)

angolo deve essere espresso in radianti.

CSNG

Converte dati numerici in numeri in precisione semplice

Sintassi
CSNG(*n*)

CVD, CVI, CVS

Convertono una variabile di tipo stringa in una variabile di tipo numerico

Sintassi
CVD (*stringa di 8 byte*)
CVI (*stringa di 2 byte*)
CVS (*stringa di 4 byte*)

CVD converte la stringa in un numero in doppia precisione, CVI in un intero e CVS in un numero in precisione semplice.

DSKF

Restituisce il numero di blocchi liberi sul disco

Sintassi
DSKF(*numdrive*)

numdrive è 0 per il drive corrente, 1 per A:, 2 per B:.

EOF

Indica la condizione di fine del file

Sintassi

EOF(*numfile*)

numfile è il numero assegnato al file al momento dell'apertura. Fornisce −1 se l'ultimo dato del file è stato letto, altrimenti 0.

EXP

Calcola la funzione esponenziale

Sintassi

EXP(*n*)

FIX

Tronca *n* ad un numero intero

Sintassi

FIX(*n*)

Elimina tutte le cifre alla destra della virgola.

FRE

Restituisce il numero di byte in memoria non utilizzati dall'MSX-BASIC

Sintassi

FRE(*n*)
FRE("*stringa*")

n e *stringa* sono argomenti fittizi.

HEX$

Restituisce una stringa che rappresenta il valore in esadecimale di un argomento decimale

Sintassi
HEX$(*n*)

Quando *n* è negativo viene restituito il suo complemento a due.

INP

Restituisce il byte letto da una porta I/O

Sintassi
INP(*porta*)

porta rappresenta l'indirizzo della porta e deve essere compreso tra 0 e 65535.

INPUT$

Restituisce una stringa di caratteri letta da tastiera o da un file

Sintassi
INPUT$(*n*[,[#] *numfile*])

n è il numero di caratteri da leggere dalla tastiera o dal file.
numfile è il numero assegnato al file al momento dell'apertura; se omesso i dati sono letti da tastiera.

INSTR

Cerca la prima occorrenza di *stringa2* in *stringa1* e restituisce la posizione in cui viene trovata l'uguaglianza

Sintassi
INSTR ([*n*,]*stringa1*, *stringa2*)

n è compreso tra 1 e 255 e assegna la posizione di partenza per la ricerca in *stringa1*.

INT

Restituisce il più grande numero intero uguale o minore di *n*

Sintassi
INT(*n*)

LEFT$

Restituisce gli *n* caratteri più a sinistra in *stringa*

Sintassi
LEFT$("*stringa*",*n*)

n deve essere un numero tra 0 e 255; se è maggiore della lunghezza di *stringa* viene restituita l'intera stringa.

LEN

Restituisce il numero di caratteri in una stringa

Sintassi
LEN("*stringa*")

Caratteri non stampabili e spazi bianchi vengono inclusi nel conteggio.

LOF

Restituisce la lunghezza di un file, cioè il numero di byte presenti nel file

Sintassi
LOF(*numfile*)

numfile è il numero assegnato al file al momento dell'apertura.

LOG

Restituisce il logaritmo naturale di *n*

Sintassi
LOG(*n*)

n deve essere maggiore di 0.

LPOS

Restituisce il valore della posizione corrente nel buffer della stampante della testina di stampa

Sintassi
LPOS(*n*)

n è un argomento fittizio.

MID$

Restituisce la parte richiesta di *stringa*

Sintassi
MID$("*stringa*", *n*[,*lunghezza*])

n è nell'intervallo da 1 a 255, *lunghezza* da 0 a 255. La funzione restituisce una stringa di *lunghezza* caratteri presa da *stringa* cominciando dal carattere *n*. Se *lunghezza* viene omessa vengono restituiti tutti i caratteri a destra di *n*.

MKD$, MKI$, MKS$

Convertono valori di tipo numerico in valori di tipo stringa

Sintassi
MKD$(*espressione in doppia precisione*)
MKI$(*espressione intera*)
MKS$(*espressione in precisione semplice*)

MKD$ converte il numero in una stringa di 8 byte; MKI$ in una stringa di 2 byte; MKS$ in una stringa di 4 byte.

OCT$

Restituisce una stringa che rappresenta il valore ottale di un argomento decimale

Sintassi
OCT$(*n*)

Se *n* è negativo viene calcolato il complemento a due.

PAD

Restituisce lo stato della tavoletta grafica

Sintassi
PAD(*n*)

n va da 0 a 3 per la porta 1 e da 4 a 7 per la porta 2.

PDL

Restituisce lo stato del paddle

Sintassi
PDL(*n*)

n va da 1 a 12; i numeri dispari per la porta A, i numeri pari per la porta B.

PEEK

Restituisce il byte letto nella locazione di memoria specificata

Sintassi
PEEK(*ind*)

ind è l'indirizzo della locazione di memoria da leggere.

PLAY

Controlla la presenza di note nel buffer musicale

Sintassi
PLAY(*n*)

n è uguale ad 1 per il canale 1, 2 per il canale 2, 3 per il canale 3. Viene restituito −1 se si trovano dati nel buffer, 0 in caso contrario.

POINT

Restituisce il codice del colore di uno specificato punto dello schermo in modo grafico

Sintassi
POINT(*x*,*y*)

(*x*,*y*) sono le coordinate del punto da verificare.

POS

Restituisce la coordinata *x* della posizione del cursore

Sintassi
POS(*n*)

n è un argomento fittizio.

RIGHT$

Restituisce il numero di caratteri specificati più a destra di *stringa*

Sintassi
RIGHT$("*stringa*",*n*)

n specifica il numero di caratteri che devono essere restituiti; se è maggiore o uguale a LEN("*stringa*") viene restituita l'intera stringa.

RND

Restituisce un numero casuale tra 0 e 1 a partire da un seme

Sintassi
RND(*n*)

Quando *n* è uguale a 0 ripete l'ultimo numero generato, quando *n* è positivo il seme resta invariato e viene generata sempre la stessa sequenza di numeri casuali, quando *n* è negativo il seme viene calcolato ogni volta sul valore di *n*.

SGN

Fornisce il segno di *n*

Sintassi
SGN(*n*)

Se *n* è positivo restituisce 1, se è 0 restituisce 0, se è negativo −1.

SIN

Restituisce il seno di *angolo*

Sintassi
SIN(*angolo*)

angolo è il valore dell'angolo in radianti.

SPACE$

Restituisce una stringa di *n* spazi bianchi

Sintassi
SPACE$(*n*)

n va da 0 a 255.

SPC

Salta *n* spazi in un'istruzione PRINT

Sintassi
SPC(*n*)

n va da 0 a 255.

SPRITE$

Contiene una stringa di caratteri che definisce la forma dello sprite

Sintassi
SPRITE$(*n*)=*stringa*

n è il numero che viene attribuito allo sprite; *stringa* è la stringa che contiene la definizione dei caratteri dello sprite.

SQR

Restituisce la radice quadrata di *n*

Sintassi
SQR(*n*)

n deve essere maggiore o uguale a zero.

STICK

Restituisce la direzione dei joystick e dei tasti di movimento del cursore

Sintassi
STICK(*n*)

Quando *n* è uguale a 0 fornisce la direzione dei tasti di movimento del cursore, quando è uguale a 1 la direzione del joystick 1, quando è uguale a 2 la direzione del joystick 2. La direzione è espressa in valori da 0 a 8 (vedi Figura 13.2).

STR$

Restituisce una rappresentazione in formato stringa del valore di *n*

Sintassi
STR$(*n*)

Se *n* è zero o positivo il primo carattere della stringa è uno spazio.

STRIG

Restituisce lo stato del pulsante del joystick o della barra di spazio

Sintassi
STRIG(*n*)

n va da 0 a 4: 0 per la barra di spazio, 1 o 3 per il joystick 1, 2 o 4 per il joystick 2. Viene restituito 0 quando non sono premuti, −1 quando uno dei due è premuto.

STRING$

Restituisce una stringa, di lunghezza definibile, i cui caratteri hanno come codice ASCII *codice* o il primo carattere di *stringa*

Sintassi
STRING$(*lunghezza,codice*)
STRING$(*lunghezza,"stringa"*)

lunghezza e *codice* devono essere compresi tra 0 e 255.

TAB

Esegue la tabulazione indicata da *n* in un'istruzione PRINT

Sintassi
TAB(*n*)

n è compreso tra 0 e 255.

TAN

Restituisce la tangente di *angolo*

Sintassi
TAN(*angolo*)

angolo è il valore dell'angolo in radianti.

USR

Richiama la subroutine indicata in codice macchina con argomento *arg*

Sintassi
USR [*n*] (*arg*)

n, che va da 0 a 9, è il valore fornito nell'istruzione DEFUSR alla routine desiderata; se viene omesso si assume USR 0; *arg* è un'espressione numerica o una variabile a stringa che diventa argomento della subroutine.

VAL

Restituisce il valore numerico di *stringa*

Sintassi
VAL("*stringa*")

Non vengono calcolati spazi bianchi, tabulazioni e a capo.

VARPTR

Restituisce l'indirizzo iniziale dove è immagazzinato un valore assegnato ad una variabile

Sintassi
VARPTR *variabile*

variabile può essere numerica, a stringa o array.

VDP

Legge o modifica i registri del chip VDP

Sintassi
VDP (*reg*)
VDP (*reg*) = *valore*

reg (da 0 a 7) è il numero del registro; *valore* va da 0 a 255.

VPEEK

Legge i dati contenuti nella memoria RAM video

Sintassi
VPEEK(*indirizzo*)

Appendice

Messaggi d'errore MSX-BASIC

La seguente lista dei possibili messaggi d'errore che vi possono giungere dall'MSX-BASIC include una breve descrizione di ogni errore. Ricordate che è possibile conoscere il numero di un errore con la variabile ERR.

Numero	Messaggio
1	**NEXT without GOSUB** Una variabile nell'istruzione NEXT non corrisponde a nessuna variabile precedentemente impiegata in una delle istruzioni FOR.
2	**Syntax error** Una linea che contiene una sequenza scorretta di caratteri (come la dimenticanza di parentesi, un comando o un'istruzione scritti male o un errore di punteggiatura). Può venire indicato il numero della linea che contiene l'errore.
3	**RETURN without GOSUB** È stata incontrata un'istruzione RETURN senza che fosse stata eseguita l'istruzione GOSUB corrispondente.
4	**Out of DATA** Un'istruzione READ è stata eseguita quando non sono rimasti da leggere dati in nessuna istruzione DATA.
5	**Illegal function call** Un parametro fuori scala è stato assegnato a un'espres-

Numero	Messaggio
	sione o a una funzione stringa. Questo errore si verifica anche nel caso di: un deponente negativo o eccessivamente grande; un argomento negativo o nullo nella funzione LOG; un argomento negativo nella funzione SQR; una mantissa negativa con un esponente non intero; una chiamata di una funzione USR per la quale l'indirizzo di partenza non è ancora stato dato; un argomento non adatto per un comando o una funzione.
6	**Overflow** Il risultato di un calcolo è troppo grande per essere rappresentato nel formato numerico dell'MSX-BASIC.
7	**Out of memory** Il programma è troppo grande, ha troppi cicli FOR o troppi GOSUB, oppure ha troppe variabili o espressioni troppo complesse.
8	**Undefined line number** La linea cui si riferisce un GOTO, un GOSUB, un IF/THEN/ELSE o un DELETE non esiste.
9	**Subscript out of range** A un elemento di un array è stato assegnato un indice che va oltre le dimensioni dichiarate, oppure un indice sbagliato.
10	**Redimensioned array** Sono state date due istruzioni DIM per lo stesso array, oppure l'istruzione DIM è stata data dopo che è già stata assunta la dimensione di default di 10 elementi.
11	**Division by zero** È stata incontrata in una espressione una divisione per zero, oppure zero è stato elevato a una potenza negativa. L'infinito con il segno del numeratore viene preso come risultato e l'esecuzione continua.
12	**Illegal direct** Un'istruzione non utilizzabile in modo diretto è stata usata come comando in modo diretto.
13	**Type mismatch** È stato assegnato un valore numerico a una variabile stringa o viceversa, oppure è stato dato un argomento stringa a una funzione che attendeva un argomento numerico o viceversa.

Numero	Messaggio
14	**Out of string space** Le variabili stringa sono andate oltre la quantità di memoria assegnata. Usate CLEAR per assegnare più spazio alle stringhe o per diminuire le dimensioni e il numero delle stringhe.
15	**String too long** Avete cercato di creare una stringa più lunga di 255 caratteri.
16	**String formula too complex** Un'espressione di tipo stringa è troppo lunga o troppo complessa. L'espressione va spezzata in espressioni più corte.
17	**Can't continue** Avete tentato di far continuare con CONT un programma interrotto da un errore o che è stato modificato durante la sosta o che non esiste.
18	**Undefined user function** Una funzione FN è stata richiamata prima che ne venisse data la definizione con un'istruzione DEF FN.
19	**Device I/O error** Un errore di I/O avviene su una periferica. È un errore "fatale" nel senso che l'MSX-BASIC non può rimediare questo errore.
20	**Verify error** Il programma in memoria è diverso da quello salvato sulla cassetta.
21	**No RESUME** Siete entrati in una routine di intercettamento degli errori (ON ERROR) ma questa non contiene l'istruzione RESUME.
22	**RESUME without error** È stata incontrata un'istruzione RESUME prima di entrare in una routine di intercettamento degli errori.
23	**Unprintable error** Non è disponibile un messaggio di errore per questa condizione d'errore. Questa è generalmente provocata da un ERROR con un codice d'errore non definito.

Numero	Messaggio
24	**Missing operand** Un'espressione contiene un operatore non seguito da un operando.
25	**Line buffer overflow** Avete tentato di immettere una linea con più di 255 caratteri.
50	**FIELD overflow** Un'istruzione FIELD sta tentando di occupare più byte di quanti ne possa occupare il record di un file ad accesso casuale
51	**Internal error** È avvenuto all'interno dell'MSX-BASIC un errore di funzionamento. Riferite al vostro rivenditore le condizioni sotto le quali si è verificato l'errore.
52	**Bad file number** Un'istruzione o un comando indica un file con un numero che non è stato dichiarato oppure che è al di fuori della gamma di numeri specificata con il comando MAXFILES=.
53	**File not found** Un'istruzione LAOD, KILL o OPEN indica un file che non esiste sul dischetto scelto.
54	**File already open** È stata data un'istruzione OPEN per output sequenziale, per un file già aperto, oppure è stato dato un comando KILL per un file aperto.
55	**Input past end** Un'istruzione INPUT si riferisce ad un file vuoto, oppure viene eseguita dopo che tutti i dati del file sono stati estratti. Per evitare questo errore usate la funzione EOF per individuare la fine del file.
56	**Bad file name** È stata usata una forma scorretta per il nome di un file in un LOAD, un SAVE, un KILL o un OPEN.
57	**Direct statement in file** Il comando LOAD ha incontrato un comando in modo diretto in un file in formato ASCII. Il comando LOAD viene bloccato.

Numero	Messaggio
58	**Sequential I/O only** È stata usata un'istruzione PUT o GET per un file sequenziale.
59	**File not OPEN** Il file indicato dalle istruzioni di input o di output (come PRINT# e INPUT#) non era stato aperto.
60-255	**Unprintable error** Questi errori non hanno messaggi.

Appendice

Guida rapida MSX-DOS

In questa appendice sono elencati i comandi MSX-DOS in ordine alfabetico. Per ognuno è indicata la sintassi e la funzione.

BASIC

Passa il controllo all'MSX-BASIC

Sintassi
BASIC [*specfile*]

Se indicato, *specfile* viene caricato ed eseguito dopo l'avviamento dell'MSX BASIC.

COPY

Esegue la copia e il concatenamento di file

Gruppo funzionale
Gestione dei file

Sintassi
COPY [{ / A| / B}] *specfile1* [{ / A| / B}] {d:|*specfile2*} [{ / A| / B}] [/ V]
COPY [{ / A| / B}] *specfile1a* [{ / A| / B}] {+*specfile1b*} [{ / A| / B}] [/ V]
COPY {*specfile1*|*perif1*} {*specfile2*|*perif2*}

DATE

Imposta la data di sistema

Gruppo funzionale
Messa a punto del sistema

Sintassi
DATE [*gg-mm-aa*]

DEL

Cancella uno o più file dal disco

Gruppo funzionale
Gestione dei file

Sintassi
DEL [*specfile*]

DIR

Visualizza il catalogo dei file presenti su disco

Gruppo funzionale
Gestione dei dischi

Sintassi
DIR [*specfile*] [/ P] [/ W]

Senza argomenti lista tutti i file presenti sul dischetto. / P visualizza il catalogo a pagine; / W visualizza solo i nomi dei file.

ERASE

Cancella uno o più file dal disco

Gruppo funzionale
Gestione dei file

Sintassi
ERASE [*specfile*]

FORMAT

Inizializza i dischi

Gruppo funzionale
Gestione dei dischi

Sintassi
FORMAT

MODE

Imposta la larghezza dello schermo

Gruppo funzionale
Messa a punto del sistema

Sintassi
MODE(*larghezza*)

larghezza è il numero massimo di colonne; deve essere compreso tra 1 e 40.

PAUSE

Ferma momentaneamente l'esecuzione di un file batch

Gruppo funzionale
Comandi per file batch

Sintassi
PAUSE [*commento*]

L'eventuale *commento* compare prima del messaggio "Strike a key when ready..."

REM

Permette di inserire stringhe di commento in un file batch

Gruppo funzionale
Comandi per file batch

Sintassi
REM [*commento*]

REN

Cambia il nome ad un file

Gruppo funzionale
Gestione dei file

Sintassi
REN *specfile1 specfile2*

specfile1 è il nome che si vuole cambiare in *specfile2*.

RENAME

Cambia il nome ad un file

Gruppo funzionale
Gestione dei file

Sintassi
RENAME *specfile1 specfile2*

specfile1 è il nome che si vuole cambiare in *specfile2*.

TIME

Visualizza e imposta l'ora

Gruppo funzionale
Messa a punto del sistema

Sintassi
TIME [*hh*[: *mm*[:*ss*]]]

Il comando è valido solo se il computer è dotato di un orologio interno.

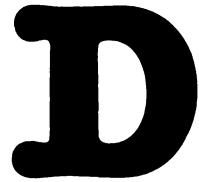
TYPE

Visualizza il contenuto di un file

Gruppo funzionale
Gestione dei file

Sintassi
TYPE *specfile*

Appendice

Codice ASCII dei caratteri

La tabella seguente mostra i caratteri corrispondenti ai vari numeri del codice ASCII. Notate che alcuni dei caratteri grafici non vengono riprodotti bene sullo schermo; questo avviene perché essi sono destinati ad uno schermo a 32 colonne. Se date il comando

```
SCREEN 1
```

quei caratteri appaiono esattamente come nella tabella. Per stampare la maggior parte dei caratteri potete usare il comando

PRINT CHR$(*n*)

dove *n* rappresenta il codice ASCII del carattere. Per esempio, per stampare il carattere alfa dell'alfabeto greco (codice ASCII 224), date il comando

```
PRINT CHR$(224)
```

È più complicato invece, stampare i caratteri che hanno un codice ASCII che va da 1 a 31. Per fare ciò dovete prima stampare il carattere con il codice ASCII 1 e poi stampare il carattere desiderato usando il suo codice ASCII accresciuto di 64. Tutto ciò si spiega meglio con un esempio. Per stampare il simbolo di cuore, che ha 3 come codice ASCII, date il comando

```
PRINT CHR$(1); CHR$(3+64);
```

Notate bene che questo vale solo per i caratteri che hanno un codice ASCII tra 1 e 31.
Il carattere 0 non stampa nulla e il carattere 255 stampa qualsiasi cosa si trovi sotto il cursore in quel momento.

Codice ASCII dei caratteri

Numero	Carattere	Numero	Carattere
1	☺	24	┌
2	☻	25	┐
3	♥	26	└
4	♦	27	┘
5	♣	28	╳
6	♠	29	╱
7	•	30	╲
8	◘	31	┼
9	○	32	
10	◙	33	!
11	♂	34	"
12	♀	35	#
13	♪	36	$
14	♫	37	%
15	☼	38	&
16	┼	39	'
17	┴	40	(
18	┬	41	)
19	┤	42	*
20	├	43	+
21	┼	44	,
22	│	45	E
23	─	46	.

Codice ASCII dei caratteri (*continua*)

Numero	Carattere	Numero	Carattere
47		70	F
48	0	71	G
49	1	72	H
50	2	73	I
51	3	74	J
52	4	75	K
53	5	76	L
54	6	77	M
55	7	78	N
56	8	79	O
57	9	80	P
58	:	81	Q
59	;	82	R
60	<	83	S
61	=	84	T
62	>	85	U
63	?	86	V
64	@	87	W
65	A	88	X
66	B	89	Y
67	C	90	Z
68	D	91	[
69	E	92	\

Codice ASCII dei caratteri (*continua*)

Numero	Carattere	Numero	Carattere
93	]	116	t
94	^	117	u
95	_	118	v
96		119	w
97	a	120	x
98	b	121	y
99	c	122	z
100	d	123	{
101	e	124	|
102	f	125	}
103	g	126	~
104	h	127	
105	i	128	Ç
106	j	129	ü
107	k	130	é
108	l	131	â
109	m	132	ä
110	n	133	à
111	o	134	å
112	p	135	ç
113	q	136	ê
114	r	137	ë
115	s	138	è

Codice ASCII dei caratteri (*continua*)

Numero	Carattere	Numero	Carattere
139	ï	162	ó
140	î	163	ú
141	ì	164	ñ
142	Ä	165	Ñ
143	Å	166	ª
144	É	167	º
145	æ	168	¿
146	Æ	169	⌐
147	ô	170	¬
148	ö	171	½
149	ò	172	¼
150	û	173	¡
151	ù	174	<<
152	ÿ	175	>>
153	Ö	176	Ã
154	Ü	177	ã
155	¢	178	Ĩ
156	£	179	ĩ
157	¥	180	Õ
158	Pt	181	õ
159	ƒ	182	Ũ
160	á	183	ũ
161	í	184	Ĳ

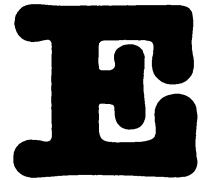
Codice ASCII dei caratteri (*continua*)

Numero	Carattere	Numero	Carattere
185	ij	208	◀
186	¾	209	⧗
187	~	210	⧓
188	◇	211	▪
189	‰	212	▪
190	¶	213	▮
191	§	214	▮
192	▬	215	▩
193	▚	216	^
194	▮	217	‡
195	▬	218	ω
196	▬	219	█
197	▮	220	▄
198	▌	221	▌
199	▞	222	▐
200	█	223	▄
201	▕	224	α
202	▐	225	β
203	⫽	226	Γ
204	⧵⧵	227	Π
205	▼	228	Σ
206	▲	229	σ
207	▶	230	μ

Codice ASCII dei caratteri (*continua*)

Numero	Carattere	Numero	Carattere
231	γ	244	⌠
232	Φ	245	⌡
233	θ	246	$\div$
234	Ω	247	$\approx$
235	δ	248	○
236	∞	249	●
237	ϕ	250	-
238	ϵ	251	$\sqrt{\ }$
239	η	252	n
240	$\equiv$	253	2
241	$\pm$	254	■
242	$\geqq$	255	
243	$\leqq$		

Appendice

Configurazione standard del sistema MSX

Le tabelle e le figure che seguono interessano soprattutto i lettori tecnicamente più esperti. Esse trattano di alcune delle parti interne dell'MSX. La Figura E.1 mostra la mappa della memoria dedicata all'MSX-DOS. La Figura E.2 mostra la mappa della memoria destinata all'MSX-BASIC. La Figura E.3 mostra la mappa della memoria riservata per l'MSX-DISK BASIC. La Tabella E.1 elenca le locazioni utili della RAM. La Tabella E.2 elenca le routine utili della ROM, che non richiedono parametri. La Tabella E.3 mostra, infine, i segnali sul bus della cartuccia e la loro direzione.

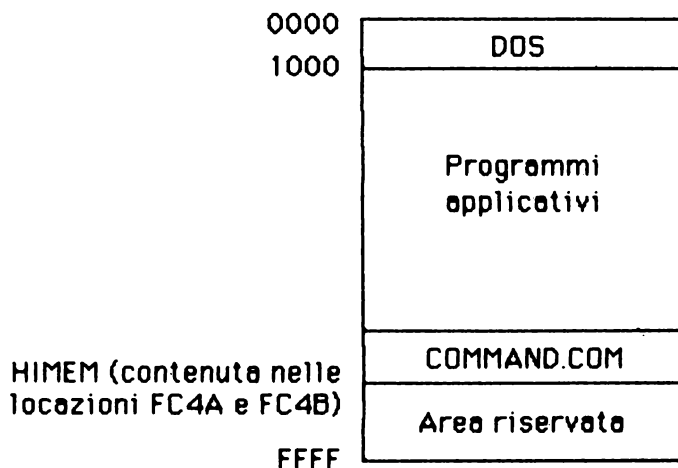



Figura E.1 Mappa della memoria dell'MSX-DOS

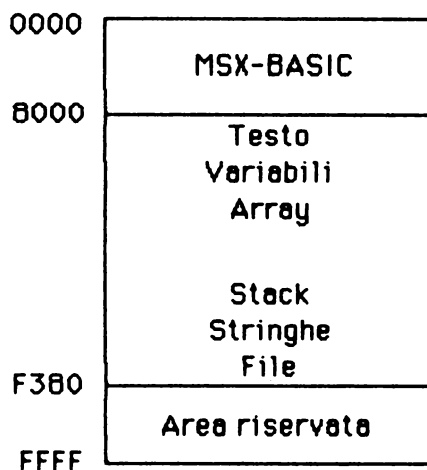



Figura E.2 Mappa della memoria dell'MSX-BASIC

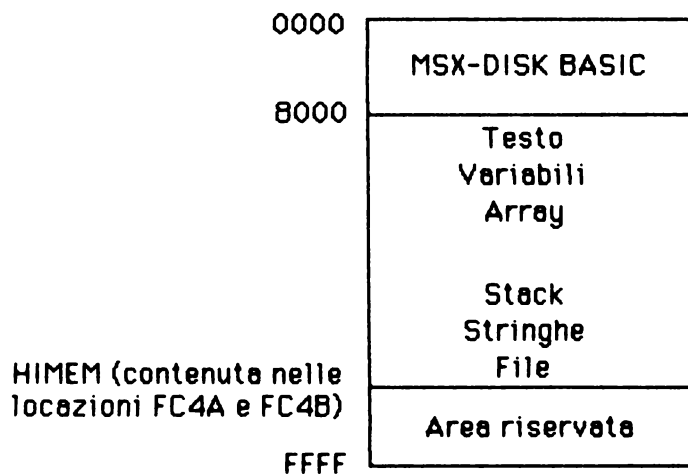



Figura E.3 Mappa della memoria dell'MSX-DISK BASIC

Tabella E.1 Locazioni utili della RAM

Indirizzo	Contenuto
F380	Routine che legge dalla porta principale
F385	Routine che scrive sulla porta principale
F39A	Indirizzo di USR0
F39C	Indirizzo di USR1...
F3AC	Indirizzo di USR9
F3B0	Lunghezza linea
F3B3	Tabella dei nomi nel modo 0
F3B5	Tabella dei colori nel modo 0
F3B7	Tabella della forma dei caratteri nel modo 0
F3B9	Tabella delle caratteristiche nel modo 0
F3BB	Tabella degli sprite nel modo 0
F3BD	Tabella dei nomi nel modo 1
F3BF	Tabella dei colori nel modo 1
F3C1	Tabella della forma dei caratteri nel modo 1
F3C3	Tabella delle caratteristiche nel modo 1
F3C5	Tabella degli sprite nel modo 1
F3C7	Tabella dei nomi nel modo 2
F3C9	Tabella dei colori nel modo 2
F3CB	Tabella della forma dei caratteri nel modo 2
F3CD	Tabella delle caratteristiche nel modo 2
F3CF	Tabella degli sprite nel modo 2
F3D1	Tabella dei nomi nel modo 3
F3D3	Tabella dei colori nel modo 3
F3D5	Tabella della forma dei caratteri nel modo 3
F3D7	Tabella delle caratteristiche nel modo 3
F3D9	Tabella degli sprite nel modo 3
F3DB	Clic dei tasti
F3DC	Coordinata *Y* del cursore
F3DD	Coordinata *X* del cursore
F3DE	Visualizzazione tasti funzione
F3E9	Colore di primo piano
F3EA	Colore dello sfondo
F3EB	Colore del bordo
F3F6	Velocità di controllo dei tasti
F3F8	Indirizzo del buffer di output della tastiera
F3FA	Indirizzo del buffer di input della tastiera
F414	Numero dell'errore
F416	Flag che attiva l'output sulla stampante
F417	Flag di controllo della stampante MSX
F418	Flag per la stampa in modo testo
F55E	Buffer della tastiera
F663	Tipo della variabile corrente
F672	Limite superiore della memoria
F676	Limite superiore dell'area delle stringhe

Tabella E.1 Locazioni utili della RAM (*continua*)

Indirizzo	Contenuto
F6A3	Puntatore ai dati
F6AA	Flag di controllo del modo AUTO
F6AB	Numero di linea corrente per AUTO
F6B3	Numero di linea dell'errore
F6B5	Numero di linea corrente
F6B9	Numero di linea di ON ERROR GOTO
F6BB	Flag di controllo di ON ERROR GOTO
F6C0	Istruzione successiva
F6C2	Limite superiore dell'area delle variabili
F6C4	Limite superiore della tabella degli array
F6C6	Fine del caricamento
F6C8	Limite superiore dell'area dati
F6CA	Specificatore di tipo delle variabili che iniziano per "A"
F6CB	Specificare di tipo delle variabili che iniziano per "B"...
F6E5	Specificatore di tipo delle variabili che iniziano per "Z"
F7BA	Numero delle funzioni definibili attive
F7C4	Flag TRON/TROFF
F922	Limite inferiore della tabella corrente del nome
F924	Limite inferiore della tabella corrente del carattere
F926	Limite inferiore della tabella corrente della forma degli sprite
F928	Limite inferiore della tabella corrente degli attributi
F931	Rapporto larghezza/altezza dell'ultimo CIRCLE (2 byte)
F93B	Numero di punti dell'ultimo CIRCLE (2 byte)
F975	Sequenza musicale della voce 1
F9F5	Sequenza musicale della voce 2
FA75	Sequenza musicale della voce 3
FC4A	Più alta locazione di memoria disponibile
FCA8	Flag del modo inserimento
FCA9	Flag di visualizzazione del cursore
FCAA	Forma del cursore
FCAB	Flag delle maiuscole
FCAF	Modo corrente dello schermo
FCB3	Coordinata *X* del cursore grafico
FCB5	Coordinata *Y* del cursore grafico

Tabella E.2 Routine della ROM che non richiedono parametri

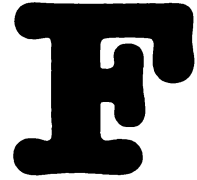
Indirizzo	Funzione
003B	Inizializza tutte le periferiche
003E	Inizializza tutti i tasti funzione
0041	Disabilita la stampa sul video
0044	Abilita la stampa sul video
005F	Modifica il colore dello schermo in base ai valori della RAM
0066	Esegue le funzioni NMI
0069	Inizializza tutti gli sprite
006C	Inizializza lo schermo in modo 0
006F	Inizializza lo schermo in modo 1
0072	Inizializza lo schermo in modo 2
0075	Inizializza lo schermo in modo 3
0078	Pone lo schermo al modo 0
007B	Pone lo schermo al modo 1
007E	Pone lo schermo al modo 2
0081	Pone lo schermo al modo 3
008A	Dà la dimensione dello sprite corrente
0090	Inizializza il suono
0099	Attiva la funzione di sfondo per PLAY
009F	Attende che venga premuto un carattere
00A8	Dà lo stato della stampante
00AE	Accetta una linea dalla tastiera
00B4	Stampa "?" e accetta una linea dalla tastiera
00BA	Controlla il tasto SHIFT-STOP
00C0	Come il BEEP dell'MSX-BASIC
00C3	Pulisce lo schermo
00CC	Cancella la visualizzazione dei tasti funzione
00CF	Visualizza i tasti funzione
00D2	Passa al modo testo
00FC	Sposta un pixel a destra
00FF	Sposta un pixel a sinistra
0102	Sposta un pixel in su
0108	Sposta un pixel in giù

Tabella E.3 Segnali sul bus della cartuccia

Pin	Nome	I/O	Descrizione
1	CS1	O	Seleziona la ROM da 4000 a 7FFF
2	CS2	O	Seleziona la ROM da 8000 a FFFF
3	CS12	O	Seleziona la ROM da 4000 a FFFF
4	SLTSL	O	Seleziona la porta
5	—		Riservato
6	RFSH	O	Refresh
7	WAIT	I	Attende dalla CPU
8	INT	I	Segnale di interrupt
9	M1	O	Riceve il segnale di accesso alla memoria della CPU
10	BUSDIR	I	Direzione del bus di dati
11	IORQ	O	Richiesta di I/O
12	MERQ	O	Richiesta di memoria
13	WR	O	Scrive
14	RD	O	Legge
15	RESET	O	Reset del sistema
16	—		Riservato
17	A9	O	Bus indirizzi
18	A15	O	" "
19	A11	O	" "
20	A10	O	" "
21	A7	O	" "
22	A6	O	" "
23	A12	O	" "
24	A8	O	" "
25	A14	O	" "
26	A13	O	" "
27	A1	O	" "
28	A0	O	" "
29	A3	O	" "
30	A2	O	" "
31	A5	O	" "
32	A4	O	" "
33	D1	I/O	Bus dati
34	D0	I/O	" "
35	D3	I/O	" "
36	D2	I/O	" "
37	D5	I/O	" "
38	D4	I/O	" "
39	D7	I/O	" "
40	D6	I/O	" "
41	GND		Terra
42	CLOCK	O	Clock della CPU
43	GND		Terra
44	SW1		Riconosce l'inserimento della cartuccia

Tabella E.3 Segnali sul bus della cartuccia (*continua*)

Pin	Nome	I/O	Descrizione
45	+5V		Alimentazione
46	SW2		Riconosce l'inserimento della cartuccia
47	+5V		Alimentazione
48	+12V		Alimentazione
49	SOUNDIN I		Input sonoro
50	−12V		Alimentazione

Appendice **F**

La tastiera

La tastiera dell'MSX è molto più completa di quanto appaia a prima vista. Infatti, ogni tasto può visualizzare ben quattro caratteri diversi. Il primo è quello che vedete segnato sul tasto, gli altri tre si ottengono mediante una combinazione dei tasti speciali CODE, SHIFT e GRAPH. Nelle figure qui di seguito sono indicati tutti i caratteri grafici e non che l'MSX può stampare.

Figura F.1 Tastiera standard

Figura F.2 Tastiera con il tasto CODE

Figura F.3 Tastiera con i tasti CODE e SHIFT

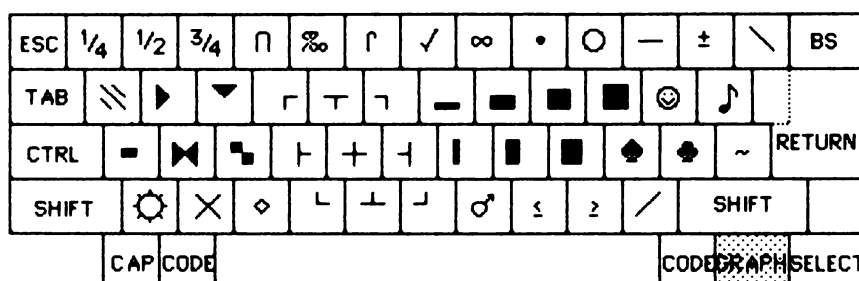



Figura F.4 Tastiera con il tasto GRAPH

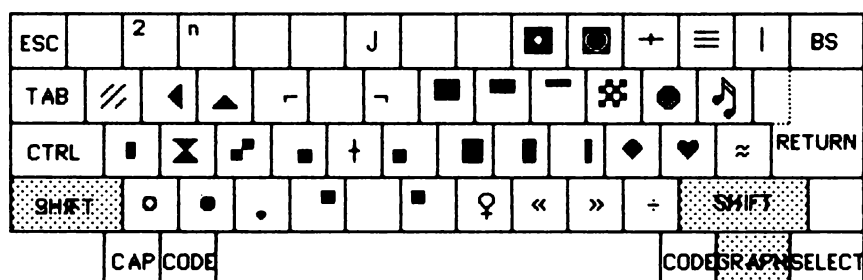



Figura E.5 Tastiera con i tasti GRAPH e SHIFT

Indice analitico

La McGraw-Hill pubblica in tutto il mondo centinaia di libri di informatica per lo studio, la professione e il tempo libero. La produzione in lingua italiana comprende:

88 7700 001 5	J. Heilborn e R. Talbott, *Guida al Commodore 64*
88 7700 002 3	C.A. Street, *La gestione delle informazioni con lo ZX Spectrum*
88 7700 003 1	T. Woods, *L'Assembler per lo ZX Spectrum*
88 7700 004 X	R. Jeffries, G. Fisher e B. Sawyer, *Divertirsi giocando con il Commodore 64*
88 7700 005 8	G. Bishop, *Progetti hardware con lo ZX Spectrum*
88 7700 006 6	H. Mullish e D. Kruger, *Il BASIC Applesoft*
88 7700 007 4	N. Williams, *Progettazione di giochi d'avventura con lo ZX Spectrum*
88 7700 008 2	H. Peckham, *Il BASIC e il PC-IBM in pratica*
88 7700 009 0	H. Peckham, *Il BASIC e il Commodore 64 in pratica*
88 7700 010 4	S. Nicholls, *Tecniche avanzate in Assembler con lo ZX Spectrum*
88 7700 011 2	K. Skier, *L'Assembler per il Commodore 64 e il VIC-20*
88 7700 012 0	S. Kamins e M. Waite, *Programmazione umanizzata in Applesoft*
88 7700 013 9	A. Pennell, *Guida allo ZX Microdrive e all'Interface 1*
88 7700 015 5	P. Cohen, *Grafica e animazione con gli Apple II*
88 7700 016 3	C. Duff, *Guida al Macintosh*
88 7700 017 1	G. Kane, *Il manuale MC68000*
88 7700 018 X	P. Hoffman e T. Nicoloff, *Il manuale MS-DOS*
88 7700 020 1	S. Nicholls, *Grafica avanzata con lo ZX Spectrum*
88 7700 021 X	L.J. Graham e T. Field, *Guida al PC-IBM*
88 7700 022 8	T. Field, *Come usare MacWrite e MacPaint*
88 7700 024 4	H. Peckham, *Il BASIC e gli Apple II in pratica*
88 7700 025 2	C. Morgan e M. Waite, *Il manuale 8086/8088*
88 7700 026 0	W. Ettlin, *Come usare il Multiplan*
88 7700 027 9	G. Mainis, *Il manuale ProDOS*
88 7700 028 7	J. Jones, *Il SuperBASIC del QL*
88 7700 029 5	C. Opie, *L'Assembler per il QL*
88 7700 030 9	W. Ettlin e G. Solberg, *Il BASIC Microsoft*
88 7700 032 5	R. Person, *Le meraviglie dell'animazione con gli Apple II*

88 7700 034 1 P. Hoffman, *Il manuale MSX*
88 7700 035 X R. Person, *Le meraviglie dell'animazione con il PC-IBM*
88 7700 036 8 W. Ettlin, *Il Multiplan per il Macintosh*
88 7700 601 3 S. Harrington, *Computer Graphics - Corso di programmazione*
88 7700 602 1 O. Lecarme e J.L. Nebut, *Pascal - Guida per programmatori*

In preparazione

88 7700 019 8 E.M. Baras, *Come usare il Simphony*
88 7700 031 7 D.L. Toppen, *Il Forth in pratica*
88 7700 038 4 L. Barnes, *Introduzione al dBase II*
88 7700 040 6 L. Barnes, *Introduzione al dBase III*
88 7700 603 X M. McGilton e R. Morgan, *Il sistema operativo UNIX*

L'MSX si è rapidamente imposto nel mercato degli home computer grazie all'elevato grado di standardizzazione tra i modelli delle varie marche e alle notevoli qualità intrinseche del sistema operativo e del BASIC.
Introduzione all'MSX è una completa guida a questo nuovo standard composto da un sistema operativo — l'MSX-DOS, derivato dal diffusissimo MS-DOS — e da un potente BASIC che non ha nulla da invidiare al famoso BASIC Microsoft, al quale è strettamente imparentato.
Dopo un'introduzione alle caratteristiche hardware e software dei sistemi MSX, attraverso una metodica descrizione di tutti i comandi, le istruzioni e le funzioni dell'MSX-BASIC, questo manuale porta l'utente, anche il più inesperto, a programmare il proprio computer e a sfruttarne le avanzate prestazioni grafiche e sonore.
L'utente più esperto, che ha a disposizione un sistema completo dotato di disk drive, troverà anche un'analitica presentazione dei comandi dell'MSX-DOS per la gestione dei file su disco e per la creazione di file batch.

Lire 27 000
(IVA 2% inclusa)

ISBN 88 7700 034 1

P. Hoffman
Il manuale MSX
McGraw Hill